A Distributed Pipelined Architecture of the Recursive Lagrangian Equations of Motion for Robot Manipulators with VLSI Implementation

by:

A.M.S. Zalzala and A.S. Morris

Department of Control Engineering,

University of Sheffield,

Mappin Street,

Sheffield S1 3JD,

U.K.

Research Report # 353

January 1989

Abstract

This paper is concerned with efficient and fast computations of robot arm joint torques, and solves the inverse dynamics problem through the design of a distributed architecture for the recursive Lagrangian equations of motion. Such recursive Lagrangian-based equations are found to be more appropriate to implement than the equivalent Newtonian-based or classical Lagrangian-based equations where trajectory planner or controller designs are involved. In addition, a symbolic representation for parts of the relevant equations are developed as a contribution to the reduction of complexity. As a result, the utilization of each element in the multiprocessor system is high. The proposed architecture is shown to be adequate for real-time applications and computationally efficient. Real time implementation is presented utilizing an INMOS T-800 transputer network, for which programming is performed in the Occam programming language, and simulation results are reported for a PUMA 560 robot manipulator, with 6 degrees-of-freedom. The practical system constructed also provided the opportunity to investigate the significance of the inter-processors communications.

I: Introduction

The problem of robot control has always presented great difficulties. In each sampling period [Brady et al 1982], a significant computational burden is involved if the robot manipulator is to be directed along a defined trajectory. A trajectory planner has the job of describing specific manipulator motion, and generating a suitable space curve for the robot end-effector to traverse. A time history of positions, velocities, accelerations and torques should be supplied to drive the control loops on the robot joints. Hence, an accurate modelling of such a planner is required to maintain correct motion. In undertaking such a task, kinematics equations are considered [Paul et al 1981; Bazerghi et al 1984; Lee and Ziegler 1984] in addition to the dynamic equations of motion [Vukobratovic and Stokic 1983; Paul 1981; Fu et al 1987] to achieve the required accuracy. However, the dynamic equations for a robot manipulator is known to be a heavy computational burden, and is extremely difficult to approach in real-time [Orin 1984; Luh et al 1980].

Tracing the history of the dynamic computations development, the first step was made by [Uicker 1965; Kahn 1969], who introduced the *classical Lagrangian* formulation. Successful attempts have been made to simplify the $O(n^4)$ time complexity of the Lagrangian formulation by introducing different levels of recursion by [Waters 1979] and later [Hollerbach 1980], yielding a reduced complexity of $O(n^2)$ and $O(n)$, respectively. Another approach using the *Newton-Euler* formulation was presented by [Luh et al 1980] yielding a $O(n)$ complexity as well. Both the Lagrangian and the Newton-Euler formulations were shown to be of equivalent origins in [Silver 1982].

In addition, one trend of simplification applied to the existing formulations is *ignoring the associated coriolis and centrifugal forces terms* , as these involve the greatest computational burden [Paul 1972,1981; Bejczy 1974]. This approximation was made under the assumption that the ignored terms were of no significance at low-speed motion, and that the resultant errors could be corrected by feedback [Luh et al 1980]. However, the acceleration-dependent terms in the dynamics equations (i.e. the coriolis and centrifugal forces terms) have been shown to be of equal importance to other velocity-dependent term at both high and low speeds [Hollerbach 1984]. Also, errors resulting from ignoring such terms cannot be easily corrected by feedback [Horn and Raibert 1978].

Aside from the computational approaches described, one different attempt was made by introducing a *table-lookup* method [Waters 1979; Albus 1975a,1975b; Raibert 1977;Horn and Raibert 1978]. The latter approach has the disadvantage of hardware

availability and data entry; in addition, there are doubts about the accuracy of this method.

For most available industrial robots, a sampling rate of $60Hz$ (or faster) is expected to achieve good real-time control [Paul 1981; Brady et al 1982; Luh and Lin 1982]. Thus, the dynamic computations must be completed within a sampling period of 1 to 15 milliseconds [Vukobratovic et al 1988]. Therefore, it is of great importance to cut down the computational burden involved if such a sampling rate is to be met.

Recently, the principles of *parallel/pipelined processing* has been introduced to simplify the computational task [Nigam and Lee 1985; Binder and Herzog 1986]. Several parallel implementations had been introduced for the Newton-Euler formulation [Luh and Lin 1982; Lathrop 1982,1985; Kasahara and Narita 1984,1985; Lee and Chang 1986; Chen et al 1988] and the Lagrangian formulation [Vukobratovic et al 1988].

The Newton-Euler formulation has linear time complexity, which is attractive for real-time applications. However, the use of these equations are not suitable for the purpose of trajectory planning and control [Sahar and Hollerbach 1986; Zalzala and Morris 1988,1989], since the torque values are given as one mixed term and not as separate variable-dependent terms [Luh et al 1980]. Hence, a trajectory time-scaling procedure [Hollerbach 1984; Lin and Chang 1985] could not be applied [Sahar and Hollerbach 1986]. One solution was offered by [Turny et al 1981], where a strobing technique was used to extract the velocity and acceleration dependent terms of the total torque value. However, such a technique has a complexity of $O(n^3)$, and hence is not suitable for real-time implementation.

The recursive Lagrangian equations of motion have a time complexity of $O(n)$. Nevertheless, they require more computations than the Newton-Euler equations [Silver 1982]. Two parallel architectures had been introduced recently for this formulation in [Lathrop 1982,1985] and [Khosla and Ramos 1988], but both of these require greater hardware resources than for the N-E formulation.

This work describes the development of a distributed formulation of the recursive Lagrangian equations of motion. Such recursive Lagrangian-based equations are found to be more appropriate to implement than the equivalent Newtonian-based or the classical Lagrangian-based equations where trajectory planners and controllers designs are involved. The proposed architecture is shown to be adequate for real-time applications, and is found to be computationally efficient. In addition, a VLSI implementation has been performed utilizing a network of INMOS T-800 transputers. This is the

first time that such a parallel processing scheme for the recursive Lagrangian equations of motion for robot manipulators has been implemented on an actual multiprocessor system.

The work will be presented as follows: In sections II and III, the theory of the recursive Lagrangian dynamics equations and the concept of concurrent programming are illustrated, respectively. In section IV, the proposed distributed formulation is discussed, while the job-scheduling aspects are tackled in section V. The performance of the proposed method is evaluated in section VI, and the implementation of the multiprocessor system along with the results of a case study are included in section VII. Finally, Conclusions are drawn in section VIII.

II: The Recursive Lagrangian Formulation

The dynamic equations of motion for robot manipulators have the characteristics of being coupled and highly nonlinear [Luh et al 1980; Paul 1981], and are expressed in terms of positions, velocities and accelerations of each joint. The original classical Lagrangian formulation was derived in [Uicker 1965; Kahn 1969], where the generalized torques/forces for an open-loop kinematic chain was defined as

$$\tau_i = \sum_{j=i}^{n}\left[\sum_{k=1}^{j}\left(tr\left[\frac{\partial \mathbf{W}_j}{\partial \theta_i} J_j \frac{\partial \mathbf{W}_j^T}{\partial \theta_k}\right]\right)\ddot{\theta}_k + \sum_{k=1}^{j}\sum_{l=1}^{j}\left(tr\left[\frac{\partial \mathbf{W}_j}{\partial \theta_i} J_j \frac{\partial^2 \mathbf{W}_j^T}{\partial \theta_k \partial \theta_l}\right]\dot{\theta}_k \dot{\theta}_l\right) - m_j \mathbf{g}^T \frac{\partial \mathbf{W}_j}{\partial \theta_i}\mathbf{r}_j\right]$$

$$, \quad i=1,2,...,n \tag{1}$$

where,

$$\mathbf{J}_j \equiv \textit{inertia tensor of link } j \text{ ,}$$

$$m_j \equiv \textit{mass of link } j \text{ ,}$$

$$\mathbf{r}_j \equiv \textit{coordinate of the centre of mass of link } j \text{ ,}$$

$$\mathbf{g} \equiv \textit{the gravity vector} \text{ ,}$$

and T and tr symbols denote the transpose and trace operations, respectively, while n is the number of degrees-of-freedom for a specific manipulator.

The various terms of (*eqn.*1) were defined as:

$$\frac{\partial \mathbf{W}_j}{\partial \theta_k} = \mathbf{W}_{k-1} \frac{\partial \mathbf{A}_k}{\partial \theta_k} \mathbf{W}_j \quad , k \leq j \tag{2}$$

$$\frac{\partial^2 \mathbf{W}_j}{\partial \theta_k \partial \theta_l} = \mathbf{W}_{k-1} \frac{\partial \mathbf{A}_k}{\partial \theta_k} \mathbf{W}_{l-1} \frac{\partial \mathbf{A}_l}{\partial \theta_l} \mathbf{W}_j \quad , k < l \leq j \tag{3}$$

$$\frac{\partial^2 \mathbf{W}_j}{\partial \theta_k \partial \theta_l} = \mathbf{W}_{k-1} \frac{\partial^2 \mathbf{A}_k}{\partial \theta_k^2} \mathbf{W}_j \quad , k = l \tag{4}$$

where a 4*x*4 transformation matrix between coordinate systems was employed in the formulation [Denavit and Hartenberg 1955], which is defined as

$$\mathbf{A}_i = \begin{bmatrix} Cos\theta_i & -Sin\theta_i Cos\phi_i & Sin\theta_i Sin\phi_i & a_i Cos\theta_i \\ Sin\theta_i & Cos\theta_i Cos\phi_i & -Cos\theta_i Sin\phi_i & a_i Sin\theta_i \\ 0 & Sin\phi i & Cos\phi_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where θ_i , ϕ_i , a_i and d_i are the arm link coordinate parameters.

The Uicker/Kahn formulation shown has an overall complexity of $O(n^4)$, and is considered to be impractical for real-time applications.

Equation (1) was partially expressed in a recursive form by [Waters 1979] giving

$$\tau_i = \sum_{j=i}^{n} \left[tr \left(\frac{\partial \mathbf{W}_j}{\partial \theta_i} J_j \ddot{\mathbf{W}}_j^T \right) - m_j \mathbf{g}^T \frac{\partial \mathbf{W}_j}{\partial \theta_i} \mathbf{r}_j \right] \tag{6}$$

where efficient computations of the coriolis and centrifugal terms lead to a reduced complexity of $O(n^2)$.

One further level of recursion was introduced by [Hollerbach 1980], where a full recursive formulation was introduced, having a linear computational complexity of $O(n)$. Nevertheless, although reaching an n dependent complexity, a further significant reduction in the number of computations had been made by utilizing $3x3$ rotational matrices rather than $4x4$ rotational-translational matrices [Hollerbach 1980]. Thus the following recursive formula was given

$$\tau_i = tr \left(\frac{\partial \mathbf{W}_i}{\partial \theta_i} \sum_{j=i}^{n} \mathbf{W}_j J_j \ddot{\mathbf{W}}_j^T \right) - g^T \frac{\partial \mathbf{W}_i}{\partial \theta_i} \sum_{j=i}^{n} (m_j \mathbf{W}_j \mathbf{r}_j) \tag{7}$$

or,

$$\tau_i = tr \left(\frac{\partial \mathbf{W}_i}{\partial \theta_i} \mathbf{D}_i \right) - \mathbf{g}^T \frac{\partial \mathbf{W}_i}{\partial \theta_i} \mathbf{c}_j \tag{8}$$

where,

$$\mathbf{D}_i = \mathbf{A}_{i+1}\mathbf{D}_{i+1} + \mathbf{p}_{i+1}\mathbf{e}_{i+1} + \frac{\mathbf{r}_i^*}{m_i}\ddot{\mathbf{p}}_i^T + \mathbf{J}_i\ddot{\mathbf{W}}_i^T \tag{9}$$

$$\mathbf{c}_i = m_i\mathbf{r}_i + \mathbf{A}_{i+1}\mathbf{c}_{i+1} \tag{10}$$

$$\mathbf{e}_i = \mathbf{e}_{i+1} + m_i\ddot{\mathbf{p}}_i^T + \left[\frac{\mathbf{r}_i^*}{m_i} \right]^T \ddot{\mathbf{W}}_i^T \tag{11}$$

$$\ddot{\mathbf{p}}_i = \ddot{\mathbf{p}}_{i-1} - \ddot{\mathbf{W}}_i\mathbf{p}_i^* \tag{12}$$

$$\mathbf{W}_i = \mathbf{W}_{i-1} \cdot \mathbf{A}_i \tag{13}$$

$$\dot{\mathbf{W}}_i = (\dot{\mathbf{W}}_{i-1} + \mathbf{W}_{i-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_i) \cdot \mathbf{A}_i \tag{14}$$

$$\ddot{\mathbf{W}}_i = \left[\ddot{\mathbf{W}}_{i-1} + 2\dot{\mathbf{W}}_{i-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_i + \mathbf{W}_{i-1} \left(\mathbf{Q}_z^2 \dot{\theta}_i^2 + \mathbf{Q}_z \ddot{\theta}_i \right) \right] \cdot \mathbf{A}_i \tag{15}$$

$$\frac{\partial \mathbf{W}_i}{\partial \theta_i} = \mathbf{W}_{i-1} \cdot \mathbf{Q}_z \cdot \mathbf{A}_i \tag{16}$$

$$\mathbf{r}_i = \mathbf{W}_i \cdot \mathbf{r}_i^* \tag{17}$$

$$\mathbf{A}_i = \begin{bmatrix} Cos\theta_i & -Sin\theta_i Cos\phi_i & Sin\theta_i Sin\phi_i \\ Sin\theta_i & Cos\theta_i Cos\phi_i & -Cos\theta_i Sin\phi_i \\ 0 & Sin\phi i & Cos\phi_i \end{bmatrix} \tag{18}$$

$$\mathbf{p}_i^* = \begin{bmatrix} a_i Cos\theta_i \\ a_i Sin\theta_i \\ d_i \end{bmatrix} \tag{19}$$

$$\mathbf{J}_i = m_i \begin{bmatrix} \frac{-k_{i11}^2 + k_{i22}^2 + k_{i33}^2}{2} + r_{ix}^2 & r_{iy}\, r_{ix} & r_{ix}\, r_{iz} \\ r_{iy}\, r_{ix} & \frac{k_{i11}^2 - k_{i22}^2 + k_{i33}^2}{2} + r_{iy}^2 & r_{iy}\, r_{iz} \\ r_{ix}\, r_{iz} & r_{iy}\, r_{iz} & \frac{k_{i11}^2 + k_{i22}^2 - k_{i33}^2}{2} + r_{iz}^2 \end{bmatrix} \tag{20}$$

and,

$\mathbf{r}_i^* \equiv$ *vector from the coordinate origin to the centre of mass of the ith link* ,

$\mathbf{p}_i \equiv$ *vector from base coordinate origin to coordinate i origin* ,

$\mathbf{p}_i^* \equiv$ *vector from coordinate i–1 origin to coordinate i origin* ,

$\mathbf{K}_i \equiv$ *radii of gyration of link i* ,

also, the matrix $\mathbf{Q}_z$ is defined for a revolute joint as

$$\mathbf{Q}_z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{21}$$

Thus, two levels of recursion are designated, one as a *Backward Recursion*, which involves (*eqn.*6), and a *Forward Recursion* including (*eqn.*7). The phrases *Backward* and *Forward* are in terms of joints numbering, representing a count of 1,2,...,*n* and *n*,*n*–1,...,1 , respectively.

III: Concurrency and VLSI Structures

The application of the concept of concurrency is found to be a well-suited solution for a wide range of technical problems in areas such as weather forecasting and genetic engineering, as well as industrial automation. It is the computational complexities associated with such applications that motivated the provision of an efficient form of information processing in cost-effective means. Concurrency concepts in an application involve the presence of parallel processing and pipelining procedures, where the task's execution time is shared and/or overlapped among the simultaneously initiated programs [Hwang and Briggs 1985]. The efficiency of this is greatly superior to the classical sequential concept of Von Neumann.

Hence, a parallel algorithm should be divided into m distinct jobs, which are then distributed over a number of p processors, all running simultaneously. However, in designing such a parallel algorithm, two paremeters are of overriding importance, namely: minimization of the execution time and maximization of every processor utilization. Thus, a time scheduling procedure is appropriate to maintain the latter requirements [Hu 1982; Winston 1984].

Parallel/pipelined processing offers large increases in computational speed. This is accomplished not only by identifying appropriate algorithmic structures that help simplifying the problem, but also by utilizing special structures of hardware designs. Recent developments in VLSI technology have made many different types of powerful processing elements available at relatively low price levels.

Several structures have been proposed for the construction of a parallel configuration [Haynes et al 1982], such as the *Systolic Architectures (or Arrays)* [Kung 1982; Kung 1987], and the *Reconfigurable Processor Array* [Snyder 1982; Jesshope 1987], which were established to provide a general methodology for mapping high-level computations onto hardware structures. The main idea in a systolic architecture is the provision of a data flow to pass through the required processing element (PEs) in one or more directions, where each PE is designed to perform a specific job. This highlights the idea of data-flow computers over the conventional sequential control-flow computers, for which maximum concurrency can be exploited in such a machine, constrained only by the hardware resource availability [Hwang and Briggs 1985]. Thus, the main aim is to construct the so-called basic elements in the required procedure, where the repetitive usage of similar elements would lead to a low-cost solution. The PEs in an architecture are often interconnected to form an array, from which a two-dimentional formation can be of different types to exploit the maximum

parallelism present in a given problem.

Considering the problem of inverse dynamics formulation for robot manipulators, a two-dimentional torus array is considered as a possible implementation [Snyder 1982; Guibas et al 1979], with the vertical extension employed for the global recursive configuration, while the horizontal extension accommodates for the local distribution. This will be discussed in the following sections.

IV: **The Distributed Recursive Lagrangian (DRL)**

IV.1 *Levels of Concurrency*:

An inherent feature in the formulation considered is the presence of recursive relations, where two levels of recursion were detected as illustrated in section II. This provides a fair justification for having a pipelined implementation. In addition, each stage of the pipeline suggested would be examined thoroughly. The equations of the manipulator are to be decomposed into a set of computational modules, where each module is called a *job*. Consequently, these jobs are connected by communications links whenever applicable. These links are essential for data transfer among each job, and are imposed according to the needs of the algorithmic implementation. Thus, a pipelined array architecture would result, as was defined in section III. Several jobs could be extracted for each phase of recursion, where a total of (10) jobs were set for phase #1 (i.e. backward recursion), and 6 jobs for phase #2 (i.e. forward recursion). It should be noted that several jobs in phase #1 were extracted from the equations involved in phase #2 of the computations. For each recursive phase indicated, one job is intended to overshadow the other jobs, yielding a significant reduction in the computational time. In addition, certain jobs consisting of matrix-matrix, matrix-vector or vector-vector multiplications (or additions) will be further operationally distributed whenever needed. Thus, parallelism in the recursive Lagrangian formulation is exploited at three levels: 1) a global level, where a pipelined configuration is imposed to accomodate for the presence of recursion in the equations, 2) a local level, where the procedure at each link is divided into several concurrent jobs, and 3) a sub-local level, yielding parallelism in the operations of a certain job within each link. The proposed configuration is illustrated in graphical form for individual links in Figure (1a). Once the pipelining is included, the structure would be in the form of a two-dimensional array for an n-link manipulator, as indicated in Figure (1b).

Considering the case where the total number of jobs involved are

$$N_{total} = N^1_{phase} + N^2_{phase} \tag{22}$$

where N^1_{phase} and N^2_{phase} denotes the number of jobs in phases #1 and #2, respectively.

Thus, the total execution time for the DRL algorithm would be

$$T_{total} = T^1_{phase} + T^2_{phase} + T_{comm} \tag{23}$$

where,

$$T^1_{phase} = \underset{i=1}{\overset{N^1_{phase}}{MAX}} \left[t^i_{job} \right] \tag{24}$$

and,

$$T^2_{phase} = \underset{i=1}{\overset{N^2_{phase}}{MAX}} \left[t^i_{job} \right] \tag{25}$$

and T_{comm} is the communications overhead involved in the flow of data.

The following jobs are considered for concurrent implementation in phase #1 of the DRL algorithm:

Job #(I,1) :

$$\mathbf{W}_j = \mathbf{W}_{j-1} \cdot \mathbf{A}_j \tag{26}$$

Job #(I,2) :

$$\dot{\mathbf{W}}_j = (\dot{\mathbf{W}}_{j-1} + \mathbf{W}_{j-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_j) \cdot \mathbf{A}_j \tag{27}$$

Job #(I,3) :

$$\ddot{\mathbf{W}}_j = \left[\ddot{\mathbf{W}}_{j-1} + 2\dot{\mathbf{W}}_{j-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_j + \mathbf{W}_{j-1} \left(\mathbf{Q}_z^2 \, \dot{\theta}_j^2 + \mathbf{Q}_z \, \ddot{\theta}_j \right) \right] \cdot \mathbf{A}_j \tag{28}$$

Job #(I,4) :

$$\ddot{\mathbf{p}}_j = \ddot{\mathbf{p}}_{j-1} - \ddot{\mathbf{W}}_j \cdot \mathbf{p}^*_j \tag{29}$$

Job #(I,5) :

$$\frac{\partial \mathbf{W}_j}{\partial \theta_j} = \mathbf{W}_{j-1} \cdot \mathbf{Q}_z \cdot \mathbf{A}_j \tag{30}$$

Job #(I,6) :

$$\mathbf{e}^*_j = \left[\frac{\mathbf{r}^*_j}{m_j} \right]^T \cdot \ddot{\mathbf{W}}^T_j \tag{31}$$

Job #(I,7) :

$$\mathbf{D}^*_j = \frac{\mathbf{r}^*_j}{m_j} \cdot \ddot{\mathbf{p}}^T_j + \mathbf{J}_j \cdot \ddot{\mathbf{W}}^T_j \tag{32}$$

Job #(I,8) :

$$\mathbf{g}^* = \mathbf{g}^T \cdot \frac{\partial \mathbf{W}_j}{\partial \theta_j} \tag{33}$$

Job #(I,9) :

$$\mathbf{A}_j = \begin{bmatrix} Cos\theta_j & -Sin\theta_j Cos\phi_j & Sin\theta_j Sin\phi_j \\ Sin\theta_j & Cos\theta_j Cos\phi_j & -Cos\theta_j Sin\phi_j \\ 0 & Sin\phi j & Cos\phi_j \end{bmatrix} \tag{34}$$

Job #(I,10) :

$$\mathbf{p}_j^* = \begin{bmatrix} a_j Cos\theta_j \\ a_j Sin\theta_j \\ d_j \end{bmatrix} \tag{35}$$

IV.2 *Symbolic Representation*:

In formulating these equations [Hollerbach 1980], a significant reduction in computations was made through the reformulation of the Lagrangian dynamics in terms of 3x3 rotational matrices instead of 4x4 rotational-transational matrices. The previous approach minimiswd the degree of sparseness in the transformation matrices. Nevertheless, unnecessary presence of sparseness was detected during the computation of several terms of the relevant dynamic equations. The terms concerned were examined, and a symbolic representation was formed for each in which unnecessary multiplications and additions were eliminated. These symbolic terms are included in *Appendix* (*A*), along with the resultant reduction in the total number of calculations associated with each.

Hence, the computational burden represented by the number of multiplications and additions involved in each job could be calculated theoretically. This is illustrated for the ten proposed jobs in Table (1). The overall configuration of these jobs could be constructed for a single link as in Figure (2a), or as a global pipelined array architecture, as shown in Figure (2b).

Table (1)
Job Computations for Phase#1 of the DRL Algorithm

Job #	Term	Multiplications				Additions			
		n=1	n>1	Total	@n=6	n=1	n>1	Total	@n=6
1	$\mathbf{W}_j$	0	24	24n-24	120	0	15	15n-15	75
2	$\dot{\mathbf{W}}_j$	6	30	30n-24	156	3	21	21n-18	108
3	$\ddot{\mathbf{W}}_j$	13	47	47n-34	248	7	36	36n-29	187
4	$\ddot{\mathbf{p}}_j$	9	9	9n	54	9	9	9n	54
5	$\frac{\partial \mathbf{W}_j}{\partial \theta_j}$	0	18	18n-18	90	3	9	9n-6	48
6	$\mathbf{e}_j^*$	9	9	9n	54	6	6	6n	36
7	$\mathbf{D}_j^*$	36	36	36n	216	27	27	27n	162
8	$\mathbf{g}^*$	9	9	9n	54	6	6	6n	36
9	$\mathbf{A}_j$	4	4	4n	24	2	2	2n	12
10	$\mathbf{p}_j^*$	2	2	2n	12	0	0	0	0

Now, considering phase #2 of the DRL algorithm, only 6 concurrent jobs remain to be implemented, as follows:

Job #(II,1) :

$$\mathbf{D}_j = \mathbf{A}_{j+1}\mathbf{D}_{j+1} + \mathbf{p}_{j+1}\mathbf{e}_{j+1} + \mathbf{D}_j^* \tag{36}$$

Job #(II,2) :

$$\mathbf{e}_j = \mathbf{e}_{j+1} + \mathbf{e}_j^* + m_j^2 \cdot \ddot{\mathbf{p}}_j^T \tag{37}$$

Job #(II,3) :

$$\tau_j^* = tr\left[\frac{\partial \mathbf{W}_j}{\partial \theta_j}\,\mathbf{D}_j\right] \tag{38}$$

Job #(II,4) :

$$\mathbf{c}_j = \mathbf{A}_{j+1} \cdot \mathbf{c}_{j+1} + m_j \cdot \mathbf{W}_j \cdot \mathbf{r}_j^* \tag{39}$$

Job #(II,5) :

$$\tau_j^{**} = \mathbf{g}^* \cdot \mathbf{c}_j \tag{40}$$

Job #(II,6) :

$$\tau_j = \tau_j^* - \tau_j^{**} \tag{41}$$

IV.3 *Distributed Operations:*

For the benefit of simplifying some of the above jobs, and reducing their computational complexity, a distributed matrix multiplication procedure is utilized. This procedure is employed for the multiplication of two 3x3 matricies *A* and *B* as follows:

```
PAR_For i=1 To 3
    SEQ_For j=1 To 3
        C[i][j] = 0.0
        SEQ_For k=1 To 3
            C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

Such a procedure can be easily modified to accommodate for matrix-vector manipulations if needed. This would be applicable for jobs (II,1) and (II,4), as shown in below:

Job #(II,1) :

$$[d_i]_j = [a_{ik}]_{j+1} \cdot [d_{ki}]_{j+1} + [p_k]_{j+1} \cdot [e_k]_{j+1} + [d_i^*]_j \tag{42}$$
$$, k,i \in \left\{ 1,2,3 \right\}$$

Job #(II,4) :

$$[c_i]_j = [a_{ik}]_{j+1} \cdot [c_k]_{j+1} + m_j \cdot [w_{ik}]_j \cdot [r_k]_j \tag{43}$$
$$, k,i \in \left\{ 1,2,3 \right\}$$

Once again, the number of additions and multiplications for each of the given jobs are calculated as shown in Table (2), where the parallel multiplication procedure mentioned earlier is included. The architecture of phase #2 is presented in Figure (3).

Now that all jobs involved in both recursive phases are identified, and their computational complexities are calculated, it should be noted that for the recursive phase #1, jobs (I,4), (I,6) and (I,7) are dependent on the output of job (I,3). Hence, these three jobs (i.e. I,4-7) are delayed one level over the other jobs involved in this phase. This 1-level delay must be accounted for when the total execution time is calculated, and will be considered as an indentation at the end of the pipeline. This indentation has been made obviouse in Figure (2b).

Table (2)
Job Computations for Phase#2 of the DRL Algorithm

Job #	Term	Multiplications				Additions			
		n=1	n>1	Total	@n=6	n=1	n>1	Total	@n=6
$\overline{1}$	$\mathbf{D}_j$	0	12	12n-12	60	0	12	12n-12	60
2	$\mathbf{e}_j$	3	3	3n	18	3	6	6n-3	33
3	τ_j^*	9	9	9n	54	8	8	8n	48
$\overline{4}$	$\mathbf{c}_j$	0	6	6n-6	30	0	5	5n-5	25
5	τ_j^{**}	3	3	3n	18	2	2	2n	12
6	τ_j	0	0	0	0	1	1	1n	6

V: The Job–Scheduling Problem

The DRL algorithm was divided into several smaller jobs in section IV. However, these jobs should be distributed optimally on a given number of processors to achieve the shortest execution time. In addition, the number of processors involved should be kept as small as possible if a cost-effective solution is to be found.

The scheduling process is performed by the optimal distribution of m jobs over p processors, achieving the minimum cost possible, where the cost indicates the overall execution time in this case. This is the most important issue in the design of multiprocessor systems, thus optimality should be reached to claim credibility for the proposed system. This issue is usually discussed in the literature as the critical-path problem [Coffman 1976; Hu 1982; Aho et al 1974], where the purpose of the analysis is to identify critical jobs in order to concentrate all available resources (i.e. processors) on them, in an attempt to reduce the total finishing time [Horwitz and Sahni 1987]. The Scheduled jobs are represented as a directed graph, in which nodes represent the jobs J_i , i=1,2,...,m, and the arcs indicating links between appropriate jobs.

The Scheduling problem in question has been considered, in general, to be extremely difficult to solve [Graham 1978; Garey and Johnson 1979; Coffman 1976]. Nevertheless, certain simplifications can often be made, which allow certain relaxations on the problem constraints and lead to a complete and successful solution [Hu 1961,1982; Nett 1976].

However, the problem of the dynamic control of robot manipulators is of extreme difficulty, where different selected jobs, with varying sub-execution times, along with an arbitrary number of application processors, increases the optimality-seeking scheduling complexity [Luh and Lin 1982; Kasahara and Narita 1985]. Hence such a problem has been classified as being strong NP-complete [Kasahara and Narita 1984; Aho et al 1974; Gary and Johnson 1979,1978], and a P-time approximation is concluded to be impossible [Kasahara and Narita 1984; Sahni and Horowitz 1978].

Nevertheless, if a scheduling scheme of some kind is to be applied, a weight factor (WF) should be associated with each of the jobs concerned. Hence, the weight factor WF_i of job J_i is defined as

$$WF_i = MULT_i \cdot M_{perf} + ADD_i \cdot A_{perf} \tag{44}$$

where,

$MULT_i \equiv$ *Number of multiplications envolved in the job* J_i,

$M_{perf} \equiv$ *The performance time of a multiplication operation,*

$ADD_i \equiv$ *Number of additions envolved in the job* J_i, and

$A_{perf} \equiv$ *The performance time of an addition operation.*

The values of $MULT_i$ and ADD_i could be easily calculated from the computational complexity Tables (1) and (2), yielding the number of operations concerned with only one joint. However, the values of M_{perf} and A_{perf} are both machine dependent, and are given the values of 550 *nanoseconds* and 350 *nanoseconds*, respectively. These values have been adopted from the implemented hardware data sheets, and are adequate to perform the scheduling process needed, and will be further discussed in section VII following. The weight factors for all jobs concerned are shown in Table (3).

Table (3)
Jobs Weight Factors

Job #	Weigt Factors (WF_i) (μSec)		Normalised Weight Factor (nWF_i)	
	Phase #1	Phase #2	Phase #1	Phase #2
1	18.5	10.8	0.48	0.68
2	23.9	3.8	0.62	0.24
3	38.5	7.8	1.00	0.49
4	9.2	5.1	0.24	0.32
5	13.1	2.4	0.34	0.15
6	7.1	0.4	0.18	0.03
7	29.3	-	0.76	-
8	7.1	-	0.18	-
9	2.9	-	0.08	-
10	1.1	-	0.03	-

In addition, a normalized weight factor (nWF) will be defined as

$$nWF_i = \frac{WF_i}{\underset{j=1}{\overset{m}{MAX}} (WF_j)} \tag{45}$$

which is a dimensionless value, while WF_i is in unit time.

In this paper, a heuristic scheduling scheme is introduced, which is a combination of both the First-Fit Decreasing (FFD) and the Best-Fit Decreasing (BFD) rules. The combined scheme, namely the FFD/BFD procedure, applies both mentioned methods on a predefined prioritized list of jobs. The First-Fit (FF) and the Best-Fit (BF) rules are illustrated in [Hu 1982; Graham 1978], which distributes a job to the lowest-index processor available, and the best available processor in the sense of minimum remaining time-space on it, respectively. In addition, the FFD and BFD procedures sort the given list of jobs into a decreasing form in terms of the jobs' normalized weight factors [Johnson 1973]. Two problems are generally encountered when applying the FF rule, namely: (1) after a few passes, several time gaps, g_i, would emerge within one or more processor, for which none of the remaining jobs would fit. Thus, $g_i < WF_i$, $i \in$ list of the remaining jobs. Also, (2) due to (1), searching for a suitable processor gets more difficult as the scheduling passes go on [Horowitz and Sahni 1987]. The FFD/BFD algorithm proposed is intended to overcome such problems, and provide a good quality solution to the scheduling task on hand.

Certain assumptions were imposed concerning the scheduling process, as follows:

A. All p processors employed are identical, and can operate on any of the available m jobs.

B. A running process cannot be interrupted until a job is completed (non-preemptive scheduling).

C. An initial priority list of jobs must be available.

The last assumption, (C), is usually met by the mathematical algorithm considered, since certain priorities may occur between several jobs as an inherent property of the algorithmic procedures. This is certainly the case in the DRL algorithm, where several jobs were found to be dependent on the output of others, as was illustrated in section IV.

Considering phase #1 of the DRL algorithm, the jobs concerned are shown as a directed graph in Figure (4a), where each of the jobs is represented as a node. The nodes are labled J_i , i=1,2,...,8, and each node encircles the job's normalized weight factor corresponding to the values of Table (3).

Respecting assumption (C) above, a priority list of decreasing order of these jobs is given as

$$PL^1_{phase} = \left\{ \left\{ J_2 , J_1 \right\}, J_3 , \left\{ J_7 , J_4 \right\}, \left\{ J_5 , J_8 \right\}, J_6 , J_9 , J_{10} \right\} \qquad (46)$$

where decreasing is in terms of the associated values of nWF_i. Thus, substituting in (*eqn*.46) with the normalized weight factors for each job yields

$$nWFPL^1_{phase} =$$

$$\left\{\left\{0.69\ ,\ 0.48\right\},\ 1.00\ ,\left\{0.76\ ,\ 0.24\right\},\left\{0.34\ ,\ 0.18\right\},\ 0.18\ ,\ 0.08\ ,\ 0.03\ ,\right\} \quad (47)$$

In sorting out the priority list, each sublist is considered as one elment with a weight factor equal to the sum of the enclosed weight factors.

The list of (*eqn*.46) has been imposed by the mathematics of the dynamics equations. The scheduling is performed by applying the FFD rule to the first element of the priority list (or any sublist within it), and further applying the BFD rule to arrange the rest of the jobs. Performing this procedure for the jobs of phase #1, a total number of 4 processors were needed, as shown in the Gantt chart form of Figure (4b).

The same procedure is applied to phase #2 of the DRL algorithm with a priority list of

$$PL^2_{phase} = \left\{\left\{\overline{J_4}\ ,\ \overline{J_1}\right\},\left\{\overline{J_4}\ ,\ \overline{J_1}\right\},\left\{\overline{J_4}\ ,\ \overline{J_1}\right\},\ J_3\ ,\ J_2\ ,\ J_5\ ,\ J_6\right\} \quad (48)$$

or,

$$nWFPL^2_{phase} =$$

$$\left\{\left\{0.32\ ,\ 0.68\right\},\left\{0.32\ ,\ 0.68\right\},\left\{0.32\ ,\ 0.68\right\},\ 0.49\ ,\ 0.24\ ,\ 0.15\ ,\ 0.03\right\} \quad (49)$$

Again, the number of processors needed is four, as shown in Figure (5), where J_1 and J_4 were distributed on 3 processors as $\overline{J_{1a}}$, $\overline{J_{1b}}$, $\overline{J_{1c}}$ and $\overline{J_{4a}}$, $\overline{J_{4b}}$, $\overline{J_{4c}}$, respectively, as indicated by (*eqns*.42,43).

VI: **Performance Analysis of the DRL**

In this section, the performance of the proposed DRL algorithm will be assessed in regard to its efficiency for implementation. Since the DRL involves both parallel and pipelining processing, a comparison between both sequential and parallel versions and sequential and parallel/pipelined versions of the equations will be conducted. The utilization degree of each processor will also be discussed to illustrate the efficiency of the scheduling procedure. In addition, the communication time of the proposed parallel/pipelined structure is to be investigated.

VI.1: *The computational complexities:*

The computational complexity of the DRL algorithm is to be calculated according to (*eqns*.23–25), while keeping in mind the 1-level indentation in phase #1 of the computations (section IV). Hence,

$$T^1_{phase} = (47n-34)+45 \quad multiplications + (36n-30)+36 \quad additions \tag{50}$$

and,

$$T^2_{phase} = (18n-18) \quad multiplications + (17n-17) \quad additions \tag{51}$$

Hence,

$$T_{total} = (65n-7) \quad multiplications + (53n-11) \quad additions + T_{comm} \tag{52}$$

Thus, for a 6 degrees-of-freedom revolute-joints robot manipulator (i.e. n=6),

$$T^{dof=6}_{total} = 383 \; multiplications + 307 \; additions + T_{comm} \tag{53}$$

The overall computational complexities of several formulations of the inverse dynamics equations are illustrated in Table (4), along with the DRL results obtained. For a 6 degrees-of-freedom (i.e. n=6) revolute-joints robot manipulator, the original formulation of the recursive Lagrangian by [Hollerbach 1980], employing 3x3 transformation matrices, has a complexity of 2195 *multiplications* and 1719 *additions*.

However, as was indicated by (*eqn*.53), the total complexity of the DRL algorithm is 383 *multiplications* and 310 *additions*, yielding reduction factors of 0.83 and 0.82, respectively. The data of Table (4) was extracted from [Hollerbach 1980] for comparison purposes.

Table (4) Computational Complexities of the Inverse Dynamics Formulations				
Method	Multiplications		Additions	
	Total	n=6	Total	n=6
Uicker,Kahn	$32\frac{1}{2}n^4 + 86\frac{5}{12}n^3 + 171\frac{1}{4}n^2 + 53\frac{1}{3}n - 128$	66271	$25n^4 + 66\frac{1}{3} + 129\frac{1}{2}n^2 + 42\frac{1}{3}n - 96$	51548
Waters	$106\frac{1}{2}n^2 + 620\frac{1}{2}n - 212$	7051	$82n^2 + 514n - 384$	5652
Hollerbach (4x4)	$830n-592$	4388	$675n-464$	3586
Hollerbach (3x3)	$412n-277$	2195	$320n-201$	1719
Newton-Euler	$150n-48$	852	$131n-48$	738
Horn,Raibert (Table Look-up)	$2n^3+n^2$	468	n^3+n^2+2n	264
DRL	$65n-7$	383	$53n-11$	307
DRL (Pipelined)	47	47	36	36

VI.2: *Processors utilization*:

The Utilization degree of each processor is defined as

$$U_p = \frac{Processor\ busy\text{-}time}{Total\ system\ execution\text{-}time} \tag{54}$$

Considering the normalized weight factors nWF_i illustrated in Table (3) and Figures (4a) and (5a) for all processors involved, the utilization value for both phases of the DRL algorithm are shown in Table (5), where a normalized total execution-time duration of 1.00 is considered.

The results of Table (5) indicates clearly the efficient utilization of each processor, where for phase #1 and #2 of the computations 75% of the involved processors are fully utilized, while only 25% of them are idle for 9% of the total execution time. Such a high utilization increases the algorithm's credibility, and allows for a low-cost VLSI implementation. However, noticing the fact that the utilization percentages

change for the first pass of the calculations (i.e. link 1) and for the 1-level indentation of phase #1, a demonstrating graph for each stage of the pipeline is shown in Figure (6).

Table (5)
Utilization of Processors

Processor #	Phase #1	Phase #2
1	0.91	1.00
2	1.00	1.00
3	1.00	1.00
4	1.00	0.91

VI.3: *The communications time*:

One important aspect in the design of a multiprocessor system is consideration of any communications between different processors in the network. In certain cases, inter-processor communications overhead may cause a corruption to a proposed algorithm [Hwang and Briggs 1985] if not handled carefully, causing delays and/or deadlocks. The time consumption of these data communications have been for long neglected by the robotics research literature, claiming its insignificance in implementation. However it is believed that the time delays between parallel-running processors, and specially when pipelining is involved, is significant, and should be included in the calculation of the total execution time. Nevertheless, the rate of data transfer is promptly machine-dependent, and its significance will be revealed in the practical implementation of section VII.

The inter-processor communications in the DRL formulation is inevitable due to the recursive nature of the equations. However, any passing of intermediate data is made between each processor and its immediate successor and/or predecessor only. In other words, no chain-passage structure is accommodated for in the structure. For phase #1 of the computations, the upmost communications overhead between any of the 4 processors involved is defined as 12 fps (floating-point numbers) including one $3x3$ matrix and one $3x1$ vector, while phase #2 is found to need an upmost overhead of 29 fps. These floating-point numbers counted are for each stage of the pipeline.

Thus, defining the following:

$COMM_{perf} \equiv$ *the performance time of a single floating point number transfer,*

$Trans^1_{phase} \equiv$ *number of fps transfer in one stage of phase*#1 (=12),

and,

$Trans^2_{phase} \equiv$ *number of fps transfer in one stage of phase*#2 (=29),

then the communication overhead for each phase is defined as:

$$T^1_{comm} = Trans^1_{phase} \,.\, COMM_{perf} \tag{55}$$

$$T^2_{comm} = Trans^2_{phase} \,.\, COMM_{perf} \tag{56}$$

The value of $COMM_{perf}$ is again hardware dependent, and is taken for the implemented practical system to be approximately (1.60 μseconds) for a single floating point number [INMOS 1988c], for which (*eqns*.55,56) gives

$$T^1_{comm} = 19.20 \ \mu seconds, \tag{57}$$

and,

$$T^2_{comm} = 46.40 \ \mu seconds, \tag{58}$$

respectively, for each stage of the algorithm.

Hence, the total execution time of the DRL can be calculated theoretically by considering (*eqn*.53). Thus,

$$T_{comm} = (n+1) \,.\, T^1_{comm} + n \,.\, T^2_{comm} \tag{59}$$

then,

$$\begin{aligned} T^{dof=6}_{total} &= 383 \times (550\times10^{-9}) + 307 \times (350\times10^{-9}) \\ &\quad + 7 \times (19.20\times10^{-6}) + 5 \times (46.40\times10^{-6}) \\ &= (\ 0.21 + 0.11 + 0.13 + 0.23\ ) \times 10^{-3} \ = (\ 0.32 + 0.36\ ) \times 10^{-3} \\ &= 0.68 \ milliseconds \end{aligned} \tag{60}$$

Considering the values of (eqn.60), it is noted that the communications time incorporates 53% of the total execution time, and is 1.13 times the operational time. This result proves the importance of including the communications overhead if a reliable functional algorithm is to be defined. Further discussion of this issue will be conducted in section VII following.

VI.4: *The effect of pipelining*:

Pipelining leads to one form of parallelism by overlaping the execution of the individual jobs. The pipeline structure associated with the DRL algorithm is linear in $n-1$ stages for both phases of the computations. Hence, such a system is expected to produce a single result every one clock interval t_{CLK}, defined as

$$t_{CLK} = \underset{i=1}{\overset{m}{MAX}} \left[WF_i \right] + t_{comm} \tag{61}$$

where t_{comm} is the communications time associated with one stage of the pipeline.

The speed-up S of the n-stages is given by

$$S = n \cdot \eta \tag{62}$$

where η is the efficiency of the pipeline

$$\eta = \frac{l}{n + (l-1)} \tag{63}$$

and l is the number of tasks to be completed. Hence, for a large value of l (i.e. $l \gg n$), η reaches its ideal value of 1, for which the throughput

$$tp = \frac{\eta}{t_{CLK}} \tag{64}$$

gives the ideal value of $\frac{1}{t_{CLK}}$.

Considering (eqn.61), the maximum number of operations involved in the pipelined version of the DRL has been estimated, as shown in Table (4), yielding a reduction factor of 0.98 for both multiplications and additions over the original RL formulation.

VII: **Practical VLSI Implementation**

VII.1: *Introduction to the transputer:*

The transputer is a general-purpose single-chip computer designed by INMOS Ltd. [INMOS 1987a], which is intended to be a significant contribution towards achieving fifth-generation supercomputers. This idea has been supported by the fact that a transputer can form a *node* among any number of similar devices, in whatever structure is needed to form a powerful multiprocessor system [Hill 1987; Vadher and Walker 1987]. The latest, and the most sophisticated, version available is the T800 [INMOS 1987b], for which an architectural view is shown in Figure (7) [INMOS 1988a,1988c]. The presence of an on-board floating-point unit significantly enhances the performance of the device, by allowing an execution rate of 1.5 million floating point operations per second. One other attraction is the communications structure predicted which allows each transputer to have four high-speed serial duplex channals linking to others in the network. Such a communication scheme would override the single-bus structure, and effectively relieves the system from a bottleneck occuring. The transputer design supports the concurrent Occam programming language [INMOS 1988b; May and Shepherd 1988; Pountain and May 1987], which allows in-depth exploition of parallel features.

The transputer has been proposed for the solution of several control applications [Taylor 1984; IEE Workshop 1987,1988; Hamblen 1987; Atkin and Ghee 1988] including robot manipulators [Jones 1985; Geffin et al 1987].

VII.2: *Mapping of the DRL:*

The significant features illustrated above promoted the use of the T800 transputer as the main block for the implementation of the DRL algorithm as a multiprocessor system. Each transputer in the network is considered as one PE accommodating one, or more, of the proposed jobs.

Excluding the pipelining feature, both phases of the algorithm are mapped on a 4-members transputer network, as shown in Figure (8). The bi-directional communications channels are set as appropriate to provide for data flow. However, it should be noted that communications among processors commences concurrently only at the end of each stage of the computations (i.e. one link of the arm), where all data flows are to be functional simultaneously, thus reducing the communications time to that of the worst channel. In addition, the bi-directional communications has the effect of increasing the link performace considerably. A study case has been investigated for a

revolute-joints robot, namely the PUMA 560 robot manipulator, with six degrees-of-freedom. Source codes for each PE have been programmed in Occam, where the symbolic representation of the computations included in *Appendex* (*A*) were fully maintained, excluding all but a minor number of structure instructions. The latter are believed to cause an increase in the execution time, and should, therefore, be minimized. Hence, employing the network of Figure (8), a total execution time of (2.46 *milliseconds*) could be achieved. The total execution time for each link of the PUMA manipulator is shown in Table (6) for both phases of computations, and including the communications time.

Table (6) *
Execution Times

Link #	Phase #1	Phase #2	Total
1	0.133	0.144	0.277
2	0.293	0.144	0.437
3	0.293	0.144	0.437
4	0.293	0.144	0.437
5	0.293	0.144	0.437
6	0.293	0.144	0.437

* All values in milliseconds

When computing the equation of motion of each link of the manipulator, two trigonometric functions would be performed, namely $Cos(theta_j)$ and $Sin(theta_j)$. These were not considered in the theory presented. Thus, although the utilization of processor $P1$ of phase #1 was shown to be 91% (Table (5)), an actual practical utilization of 98% was detected on the corresponding transputer.

The Communications activities between each of the four processors is illustrated in Tables (7a) and (7b) for both phases of the algorithm.

Table (7a)
Communications Activities for Phase #1

Pass #	T0		T1		T2		T3	
	Send	Receive	Send	Receive	Send	Receive	Send	Receive
0	-	-	T2 T3	-	-	T1	-	T1
1	-	T1 T2 T3	T0 T2 T3	T3	T0	T1 T3	T0 T1 T2	T1
2	-	T1 T2 T3	T0 T2 T3	T3	T0	T1 T3	T0 T1 T2	T1
3	-	T1 T2 T3	T0 T2 T3	T3	T0	T1 T3	T0 T1 T2	T1
4	-	T1 T2 T3	T0 T2 T3	T3	T0	T1 T3	T0 T1 T2	T1
5	-	T1 T2 T3	T0 T2 T3	T3	T0	T1 T3	T0 T1 T2	T1
6	-	T1 T2 T3	T0	T3	T0	-	T0 T1	-
7	-	T1	T0	-	-	-	-	-

Table (7b)
Communications Activities for Phase #2

Pass1 #	T0		T1		T2		T3	
	Send	Receive	Send	Receive	Send	Receive	Send	Receive
1	T1 T2 T3	T1 T2 T3	T0	T0	T0	T0	T0	T0
2	T1 T2 T3	T1 T2 T3	T0	T0	T0	T0	T0	T0
3	T1 T2 T3	T1 T2 T3	T0	T0	T0	T0	T0	T0
4	T1 T2 T3	T1 T2 T3	T0	T0	T0	T0	T0	T0
5	T1 T2 T3	T1 T2 T3	T0	T0	T0	T0	T0	T0
6	-	-	-	-	-	-	-	-

A noticable difference occures between the total execution time between the theoretical value obtained by (*eqn.*60) and that produced by the implemented transputer network, as shown in Table (9). This is due to the precence of delay elements in the executed code (e.g. matrix indexing and control structures) in addition to the

communications mechanism of the links.

Including the pipelining in the structure of the DRL for this case study yields a two-dimensional array architecture, for which a network of 26 transputers is needed, as shown in Figure (9). The speed-up factor associated with such an implementation is calculated from (*eqn.*62) for a 300 point trajectory, and is found to be (6.86), yielding an efficiency of $\eta = 0.98$. However, it should be mentioned that different communication configuration must be applied for this implementation. This is due to the 4-link limitation imposed upon each transputer. Thus, data must flow between two transputers through a third one in each stage of the pipeline. The speed-up values involved using a multiple of 4 transputers (*i.e.* 4,8,12,...) is shown in Figure (10). The time sequence for each stage of the pipelined array is shown in tables (8a) and (8b) for both phases of the algorithm, with a total execution time of (0.026 *milliseconds*).

Table (8a) *
Pipelined execution-time for Phase #1

Stage #	T0	T1	T2	T3
1	0.0	0.096	0.096	0.0
2	0.25	0.25	0.25	0.25
3	0.25	0.25	0.25	0.25
4	0.25	0.25	0.25	0.25
5	0.25	0.25	0.25	0.25
6	0.25	0.25	0.25	0.25
7	0.224	0.064	0.0	0.0

* All values in milliseconds

Table (8b) * Pipelined execution-time for Phase #2				
Stage #	T0	T1	T2	T3
1	0.144	0.144	0.144	0.144
2	0.144	0.144	0.144	0.144
3	0.144	0.144	0.144	0.144
4	0.144	0.144	0.144	0.144
5	0.144	0.144	0.144	0.144
6	0.144	0.0	0.0	0.0
* All values in milliseconds				

As a comparison between the T800 transputer and the older version, the T414 [INMOS 1987c], the same configuration was executed for a T414 network. The results are reported in Table (9), which clearly shows the superiority of the T800.

Table (9) * T800 vs T414 Results			
Transputer	Theoretical	Practical	Pipelined **
T800	0.68	2.46	0.026
T414	7.72	8.96	1.09
* All values in milliseconds ** Simulation results			

VIII: **Conclusion**

A distributed algorithm of the recursive Lagrangian equations of motion for robot manipulators has been introduced. These equations are believed to be more suitable for tackling the problem of robot control than the equivilant Neuton-Euler equations. The proposed algorithm was implemented on a multiprocessor system, yielding almost full utilization of the network. A practical VLSI system has been constructed, for which the importance of the inter-processors communications was demonstrated. The performance of the system is found to be close to that predicted by the theory, and is shown to be adequate for real-time robot control.

Appendix (A)

Symbolic Representation of Terms

Each matrix involved in the computation of the following terms has a general structure as follows:

$$\mathbf{M}_j = \begin{bmatrix} m^j_{11} & m^j_{21} & m^j_{31} \\ m^j_{21} & m^j_{22} & m^j_{32} \\ m^j_{13} & m^j_{23} & m^j_{33} \end{bmatrix} \tag{A.1}$$

and the index i is defined as

$$i = 1,2,3$$

whenever applicable.

(1): Evaluating $\mathbf{W}_j = \mathbf{W}_{j-1} \cdot \mathbf{A}_j$,$(j > 1)$:

$$w^j_{11} = w^{j-1}_{11} a^j_{11} + w^{j-1}_{12} a^j_{21} \tag{A.2}$$

$$w^j_{12} = w^{j-1}_{11} a^j_{12} + w^{j-1}_{12} a^j_{22} + w^{j-1}_{13} a^j_{32} \tag{A.3}$$

$$w^j_{13} = w^{j-1}_{11} a^j_{13} + w^{j-1}_{12} a^j_{23} + w^{j-1}_{13} a^j_{33} \tag{A.4}$$

$$w^j_{21} = w^{j-1}_{21} a^j_{11} + w^{j-1}_{22} a^j_{21} \tag{A.5}$$

$$w^j_{22} = w^{j-1}_{21} a^j_{12} + w^{j-1}_{22} a^j_{22} + w^{j-1}_{23} a^j_{32} \tag{A.6}$$

$$w^j_{23} = w^{j-1}_{21} a^j_{13} + w^{j-1}_{22} a^j_{23} + w^{j-1}_{23} a^j_{33} \tag{A.7}$$

$$w^j_{31} = w^{j-1}_{31} a^j_{11} + w^{j-1}_{32} a^j_{21} \tag{A.8}$$

$$w^j_{32} = w^{j-1}_{31} a^j_{12} + w^{j-1}_{32} a^j_{22} + w^{j-1}_{33} a^j_{32} \tag{A.9}$$

$$w^j_{33} = w^{j-1}_{31} a^j_{13} + w^{j-1}_{32} a^j_{23} + w^{j-1}_{33} a^j_{33} \tag{A.10}$$

* *Original Computations* = 27 *multiplications* + 18 *additions*.

* *Reduced Computations* = 24 *multiplications* + 15 *additions*.

(2): Evaluating $\dot{\mathbf{W}}_j = (\dot{\mathbf{W}}_{j-1} + \mathbf{W}_{j-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_j) \cdot \mathbf{A}_j = \mathbf{V}_j \cdot \mathbf{A}_j$:

A. j=1 :

$$\dot{w}^j_{1i} = - a^j_{2i} \cdot \dot{\theta}_j \tag{A.11}$$

$$\dot{w}^j_{2i} = a^j_{1i} \cdot \dot{\theta}_j \tag{A.12}$$

$$\dot{w}^j_{31} = 0.0 \tag{A.13}$$

* *Original Computations* = 27 *multiplications* + 18 *additions.*

* *Reduced Computations* = 6 *multiplications* + 3 *additions.*

B. j>1 :

$$v^j_{i1} = w^{j-1}_{i2} \cdot \dot{\theta}_j + \dot{w}^{j-1}_{i1} \tag{A.14}$$

$$v^j_{i2} = w^{j-1}_{i1} \cdot \dot{\theta}_j + \dot{w}^{j-1}_{i2} \tag{A.15}$$

$$v^j_{i3} = w^{j-1}_{i3} \tag{A.16}$$

then,

$$\dot{w}^j_{11} = v^{j-1}_{11} a^j_{11} + v^{j-1}_{12} a^j_{21} \tag{A.17}$$

$$\dot{w}^j_{12} = v^{j-1}_{11} a^j_{12} + v^{j-1}_{12} a^j_{22} + v^{j-1}_{13} a^j_{32} \tag{A.18}$$

$$\dot{w}^j_{13} = v^{j-1}_{11} a^j_{13} + v^{j-1}_{12} a^j_{23} + v^{j-1}_{13} a^j_{33} \tag{A.19}$$

$$\dot{w}^j_{21} = v^{j-1}_{21} a^j_{11} + v^{j-1}_{22} a^j_{21} \tag{A.20}$$

$$\dot{w}^j_{22} = v^{j-1}_{21} a^j_{12} + v^{j-1}_{22} a^j_{22} + v^{j-1}_{23} a^j_{32} \tag{A.21}$$

$$\dot{w}^j_{23} = v^{j-1}_{21} a^j_{13} + v^{j-1}_{22} a^j_{23} + v^{j-1}_{23} a^j_{33} \tag{A.22}$$

$$\dot{w}^j_{31} = v^{j-1}_{31} a^j_{11} + v^{j-1}_{32} a^j_{21} \tag{A.23}$$

$$\dot{w}^j_{32} = \dot{w}^{j-1}_{31} a^j_{12} + v^{j-1}_{32} a^j_{22} + v^{j-1}_{33} a^j_{32} \tag{A.24}$$

$$\dot{w}^j_{33} = v^{j-1}_{31} a^j_{13} + v^{j-1}_{32} a^j_{23} + v^{j-1}_{33} a^j_{33} \tag{A.25}$$

* *Original Computations* = 63 *multiplications* + 45 *additions.*

* *Reduced Computations* = 30 *multiplications* + 21 *additions.*

(3): Evaluating

$\ddot{\mathbf{W}}_j = \left[\ddot{\mathbf{W}}_{j-1} + 2\dot{\mathbf{W}}_{j-1} \cdot \mathbf{Q}_z \cdot \dot{\theta}_j + \mathbf{W}_{j-1} \left(\mathbf{Q}_z^2 \, \dot{\theta}_j^2 + \mathbf{Q}_z \, \ddot{\theta}_j \right) \right] \cdot \mathbf{A}_j = \mathbf{U}_j \cdot \mathbf{A}_j$:

A. j=1 :

$$\rho = \dot{\theta}_j \cdot \dot{\theta}_j \tag{A.26}$$

$$\ddot{w}^j_{11} = -\rho \cdot w^{j-1}_{11} - \ddot{\theta}_j \cdot w^{j-1}_{21} \tag{A.27}$$

$$\ddot{w}^j_{12} = -\rho \cdot w^{j-1}_{12} - \ddot{\theta}_j \cdot w^{j-1}_{22} \tag{A.28}$$

$$\ddot{w}^j_{13} = -\rho \cdot w^{j-1}_{13} - \ddot{\theta}_j \cdot w^{j-1}_{23} \tag{A.29}$$

$$\ddot{w}^j_{21} = \ddot{\theta}_j \cdot w^{j-1}_{11} - \rho \cdot w^{j-1}_{21} \tag{A.30}$$

$$\ddot{w}^j_{22} = \ddot{\theta}_j \cdot w^{j-1}_{12} - \rho \cdot w^{j-1}_{22} \quad (A.31)$$

$$\ddot{w}^j_{23} = \ddot{\theta}_j \cdot w^{j-1}_{13} - \rho \cdot w^{j-1}_{23} \quad (A.32)$$

$$\ddot{w}^j_{31} = \ddot{w}^j_{32} = \ddot{w}^j_{33} = 0.0 \quad (A.33)$$

* *Original Computations* = 73 *multiplications* + 45 *additions*.
* *Reduced Computations* = 13 *multiplications* + 7 *additions*.

B. j>1 :

$$\rho = \dot{\theta}_j \cdot \dot{\theta}_j \quad (A.34)$$

$$\delta = 2\,\dot{\theta}_j \quad (A.35)$$

$$u^j_{i1} = -\rho \cdot w^{j-1}_{i1} + \ddot{\theta}_j \cdot w^{j-1}_{i2} + \delta \cdot \dot{w}^{j-1}_{i2} + \ddot{w}^{j-1}_{i1} \quad (A.36)$$

$$u^j_{i2} = -\ddot{\theta}_j \cdot w^{j-1}_{i1} - \rho \cdot w^{j-1}_{i2} - \delta \cdot \dot{w}^{j-1}_{i1} + \ddot{w}^{j-1}_{i2} \quad (A.37)$$

$$u^j_{i3} = \ddot{w}^{j-1}_{i3} \quad (A.38)$$

* *Original Computations* = 137 *multiplications* + 99 *additions*.
* *Reduced Computations* = 47 *multiplications* + 36 *additions*.

(4): Evaluating $\dfrac{\partial \mathbf{W}_j}{\partial \theta_j} = \mathbf{N}_j$:

A. j=1 :

$$n^j_{1i} = -a^j_{2i} \quad (A.39)$$

$$n^j_{2i} = a^j_{1i} \quad (A.40)$$

$$n^j_{3i} = 0.0 \quad (A.41)$$

* *Original Computations* = 27 *multiplications* + 18 *additions*.
* *Reduced Computations* = 3 *additions*.

B. j>1 :

$$n^j_{i1} = -w^{j-1}_{i1} \cdot a^j_{21} + w^{j-1}_{i2} \cdot a^j_{11} \quad (A.42)$$

$$n^j_{i2} = -w^{j-1}_{i1} \cdot a^j_{22} + w^{j-1}_{i2} \cdot a^j_{12} \quad (A.43)$$

$$n^j_{i3} = -w^{j-1}_{i1} \cdot a^j_{23} + w^{j-1}_{i2} \cdot a^j_{13} \quad (A.44)$$

* *Original Computations* = 54 *multiplications* + 36 *additions*.
* *Reduced Computations* = 18 *multiplications* + 9 *additions*.

References

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D., (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

ALBUS, J. S., (1975). "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME, J. Dyn. Syst., Meas., Cont.*, vol. 97, pp. 270-277.

ALBUS, J. S., (1975). "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME, J. Dyn. Syst., Meas., Cont.*, vol. 97, pp. 228-233.

ATKIN, P. AND GHEE, S., (1988). "A Transputer Based Multi-User Flight Simulator," INMOS Technical Note #36.

BAZERGHI, A., GOLDENBERG, A. A., AND APKARIAN, J., (1984). "An Exact Kinematic Model of PUMA 600 Manipulator," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-14, pp. 483-87.

BEJCZY, A. K., (1974). "Robot Arm Dynamics and Control," *JPL Labs. Tech. Memo #33-669.*

BINDER, E. E. AND HERZOG, J. H., (1986). "Distributed Computer Architecture and Fast Parallel Algorithms in Real-Time Robot Control," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-16, no. 4, pp. 543-549.

BRADY, J. M., HOLLERBACH, J. M., JOHNSON, T. L., LOZANO-PEREZ, T., AND MASON, M. T., (1982). *Robot Motion : Planning and Control*, MIT Press.

CHEN, C. L., LEE, C. S. G., AND HOU, E. S. H., (1988). "Efficient Scheduling Algorithms For Robot Inverse Dynamics Computation on a Multiprocessor System," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, pp. 1146-1151.

COFFMAN, E. G., (1976). *Computer and Job-Shop Scheduling Theory*, NewYork:Wiley.

DENAVIT, J. AND HARTENBERG, R. S., (1955). "A Kinematic Notation for Lower-pair Mechanisms Based on Matrices," *J. App. Mech.*, vol. 77, pp. 215-221.

FU, K. S., LEE, C. S. G., AND GONZALIS,, (1987). *Robotics : Control, Sensing, Vision and intelligence*, McGraw Hill.

GAREY, M. R. AND JOHNSON, D. S., (1978). "Strong NP-Completeness Results: Motivation, Example and Implications," *J. ACM*, vol. 25, pp. 499-508.

GAREY, M. R. AND JOHNSON, D. S., (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco,CA:Freeman.

GRAHAM, R. L., (1978). "The Combinatorial Mathematics of Scheduling," *Scientific American*, vol. 238, no. 3, pp. 104-123.

GUIBAS, L. J., KUNG, H. T., AND THOMPSON, C. D., (1979). "Direct VLSI Implementation of Combinatorial Algorithms," in *Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication*, pp. 509-525.

HAYNES, L. S., LAU, R. L., SIEWIOREK, D. P., AND MIZELL, D. W., (1982). "A Survay of Highly Parallel Computing," *IEEE Computer*, vol. 15, no. 1, pp. 47-56.

HILL, G., (1988). "Transputer Networks Using the IMS B003," INMOS Technical Note #13.

HOLLERBACH, J. M., (1980). "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-10, pp. 730-36.

HOLLERBACH, J. M., (1984). "Dynamic Scaling of Manipulator Trajectories," *Trans. ASME, J. of Dyn. Syst., Meas. and Control* , vol. 106, pp. 102-06.

HORN, B. K. P. AND RAIBERT, M. H., (1978). "Manipulator Control Using the Configuration Space Method," *The Industrial Robot*, vol. 5, no. 2, pp. 69-73.

HOROWITZ, E. AND SAHNI, S., (1987). *Fundamentals of Data Structures in Pascal*, Computer Science Press.

HU, T.C., (1961). "Parallel Sequencing and Assembly Line Problems," *J. Operations Research*, vol. 9, no. 6, pp. 841-848.

HU, T. C., (1982). *Combinatorial Algorithms*, Addison-Wesley.

HWANG, K. AND BRIGGS, F. A., (1985). *Computer Architecture and Parallel Processing*, McGraw-Hill.

INMOS,(1987a), *Product Overview: The Transputer Family.*

INMOS,(1987b), *Engineering Data: IMS T800 Transputer.*

INMOS,(1987c), *Engineering Data: IMS T414 Transputer.*

INMOS,(1988a), *Transputer Development System*, Prentice-Hall.

INMOS,(1988b), *Occam-2 Reference Manual*, Prentice-Hall.

INMOS,, (1988c). "IMS T800 Transputer," Technical Note #6, Bristol.

JESSHOPE, C., (1987). "The RPA as an Intelligent Transputer Memory System in an Occam Programming Model," in *Systolic Arrays, eds. W. Moore, A. McCabe and R. Urquhart*, Adam Hilger.

JOHNSON, D. S., (1973). "Near-Optimal Bin-Packing Algorithms," *MIT report MAC TR-109.*

KAHN, M. E., (1969). "The Near-Minimum Time Control of Open-Loop Articulated Kinematic Chains," *Stanford A.I. Lab., AIM 106.*

KASAHARA, H. AND NARITA, S., (1984). "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," *IEEE Trans. Computers*, vol. C-33, no. 11, pp. 1023-1029.

KASAHARA, H. AND NARITA, S., (1985). "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System," *IEEE J. Robotics and Automation*, vol. RA-1, no. 2, pp. 104-113.

KHOSLA, P. R. AND RAMOS, S., (1988). "A Comparative Analysis of the Hardware Requirements for the Lagrange-Euler and Newton-Euler Dynamics Formulations," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, pp. 291-296.

KUNG, H. T., (1982). "Why Systolic Architectures?," *IEEE Computer*, vol. 15, no. 1, pp. 37-46.

KUNG, S. Y., (1987). "VLSI Array Processors," in *Systolic Arrays, eds. W. Moore, A. McCabe and R. Urquhart*, Adam Hilger.

LATHROP, R. H., (1982). "Parallelism in Arms and Legs," *S.M. Thesis, MIT.*

LATHROP, R. H., (1985). "Parallelism in Manipulator Dynamics," *Int. J. Robotics Research*, vol. 4, no. 2, pp. 80-102.

LEE, C.S.G. AND ZIEGLER, M., (1984). "A Geometric Approach in Solving The Inverse Kinematics of PUMA Robot," *IEEE Trans.* , vol. AES-20, no. 6, pp. 695-706.

LEE, C. S. G. AND CHANG, P. R., (1986). "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-16, no. 4, pp. 532-542.

LIN, C. S. AND CHANG, P. R., (1985). "Approximate Optimum Paths of Robot Manipulators Under Realistic Physical Constraints," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 737-42.

LUH, J. Y. S., WALKER, M. W., AND PAUL, R. P. C., (1980). "On-Line Computational Scheme for Mechanical Manipulators," *Trans. ASME, J. of Dyn. Syst., Meas. and Control*, vol. 102, pp. 69-76.

LUH, J. Y. S. AND LIN, C. S., (1982). "Scheduling of Parallel Computation for a Computer Controlled Mechanical Manipulator," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-12, no. 2, pp. 214-234.

MAY, D. AND SHEPHERD, R., (1988). "The Transputer Implementation of Occam," INMOS Technical Note #21.

NETT, E., (1976). "On Further Application of the Hu Algorithm to Scheduling Problems," in *Proc. Int. Conf. Parallel Processing, P.H. Enslow, Jr. (ed.).*

NIGAM, R. AND LEE, C. S. G., (1985). "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators," *IEEE J. Robotics Automation*, vol. 1, no. 4, pp. 173-182.

ORIN, D. E., (1984). "Pipelined Approach to Inverse Plant Plus Jacobian Control of Robot Manipulators," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 169-175.

PAUL, R. P., (1972). "Modeling, Trajectory Calculation and Servoing of a Computer Controlled Arm'," *Ph.D. Dissertation, Stanford University.*

PAUL, R. P., (1981). *Robot Manipulators : Mathmatics, Programming and Control*, MIT Press.

PAUL, R. P., SHIMANO, B., AND MAYER, G. E., (1981). "Kinematic Control Equations for Simple Manipulators," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-11, pp. 449-55.

POUNTAIN, D. AND MAY, D., (1987). *A Tutorial Introduction to Occam Programming*, BSP Professional Books.

RAIBERT, M. H., (1977). "Analytical Equation vs. Table Look-up for Manipulation: A Unifying Concept," in *Proc. IEEE Conf. Decision and Control*, pp. 576-579.

SAHAR, G. AND HOLLERBACH, J. M., (1986). "Planning of Minimum-time Trajectories for Robot Arms," *Int. J. Robotics Research*, vol. 5, no. 3, pp. 90-100.

SAHNI, S. AND HOROWITZ, E., (1978). "Combinatorial Problems Reducibility and Approximation," *J. Operations Research*, vol. 26, pp. 718-759.

SILVER, W., (1982). "On the Equivilance of Lagrangian and Newton-Euler Dynamics for Manipulators," *Int. J. Robotics Research*, vol. 1, no. 2, pp. 60-70.

SNYDER, L., (1982). "Introduction to the Configurable, Highly Parallel Computer," *IEEE Computer*, vol. 15, no. 1, pp. 47-56.

TAYLOR, R., (1984). "Signal Processing With Occam and the Transputer," *IEE Proc.*, vol. 131-F, no. 6, pp. 610-614.

TURNEY, J. L., (1981). "Connection Between Formulations of Robot Arm Dynamics with Applications to Simulation and Control," *Univ. of Michigan, Report RSD-TR-4-82.*

UICKER, J. J., (1965). "On the Dynamic Analysis of Spatial Linkages Using 4x4 Matricies," *Ph.D. Thesis, Northwestern Univ.*.

VUKOBRATOVIC, M. AND STOKIC, D., (1983). "Is Dynamic Control Needed in Robotic Systems, and, if So, to What Extent ?," *Int. J. Robotics Research*, vol. 2, no. 2, pp. 18-34.

VUKOBRATOVIC, M., KIRCANSKI, N., AND LI, S. G., (1988). "An Approach to Parallel Processing of Dynamic Robot Models," *Int. J. Robotics Research*, vol. 7, no. 2, pp. 64-71.

WALKER, P., (1987). "IMS B003 Design of a Multi-transputer Board," INMOS Technical Note #10.

WATERS, R. C., (1979). "Mechanical Arm Control," *MIT AI-Lab Memo #549.*

WINSTON, P. H., (1984). *Artificial Intelligence*, Addison-Wesley.

ZALZALA, A. M. S. AND MORRIS, A. S., (1988). "An Optimum Trajectory Planner for Robot Manipulators in Joint-Space and Under Physical Constraints," *Univ. of Sheffield, Dept. of Control Eng., Research Report #349.*

ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "Optimum Trajectory Planning for Robot Manipulators," *To appear, IMA Conf. on Robotics: Applied Mathematics and Computational Aspects, 12-14 July*, Loughborough Univ. of Technology, U.K..

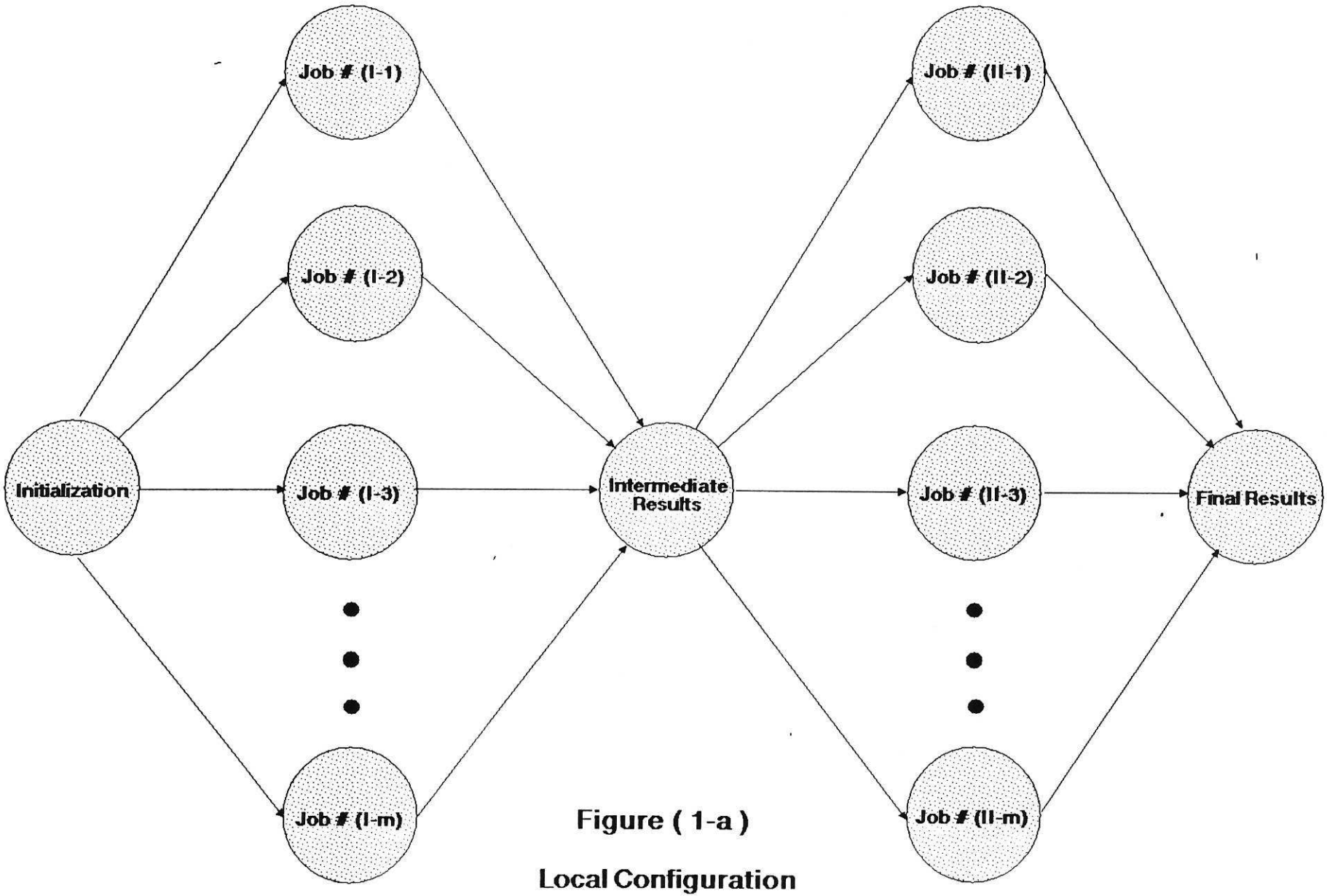


Figure (1-a)

Local Configuration
of the DRL Algorithm

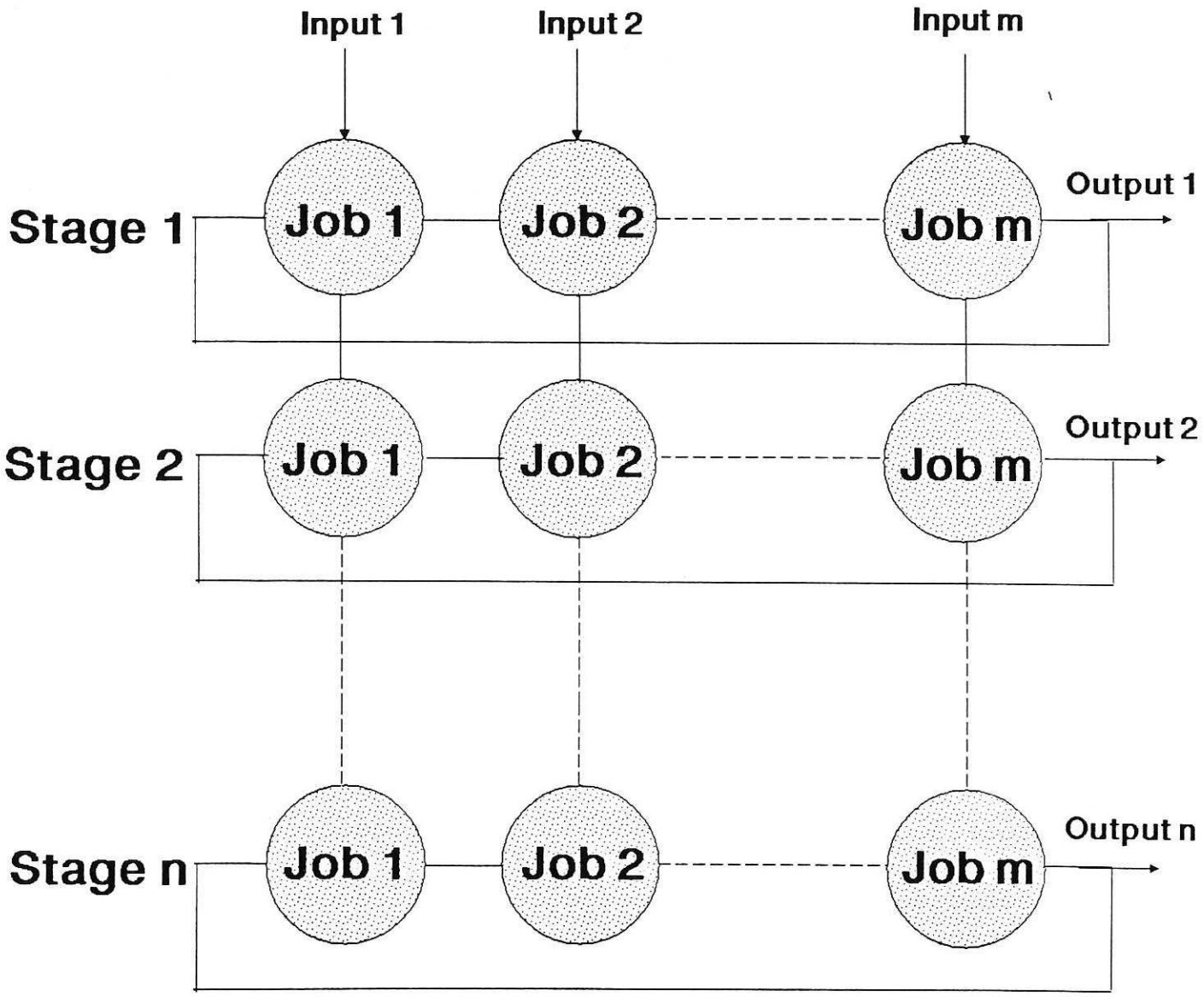
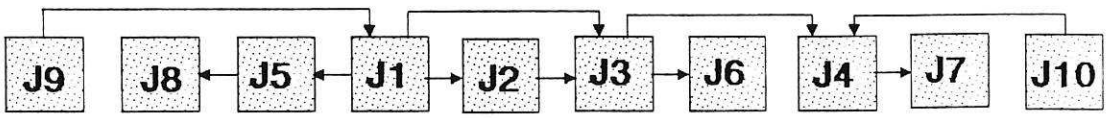
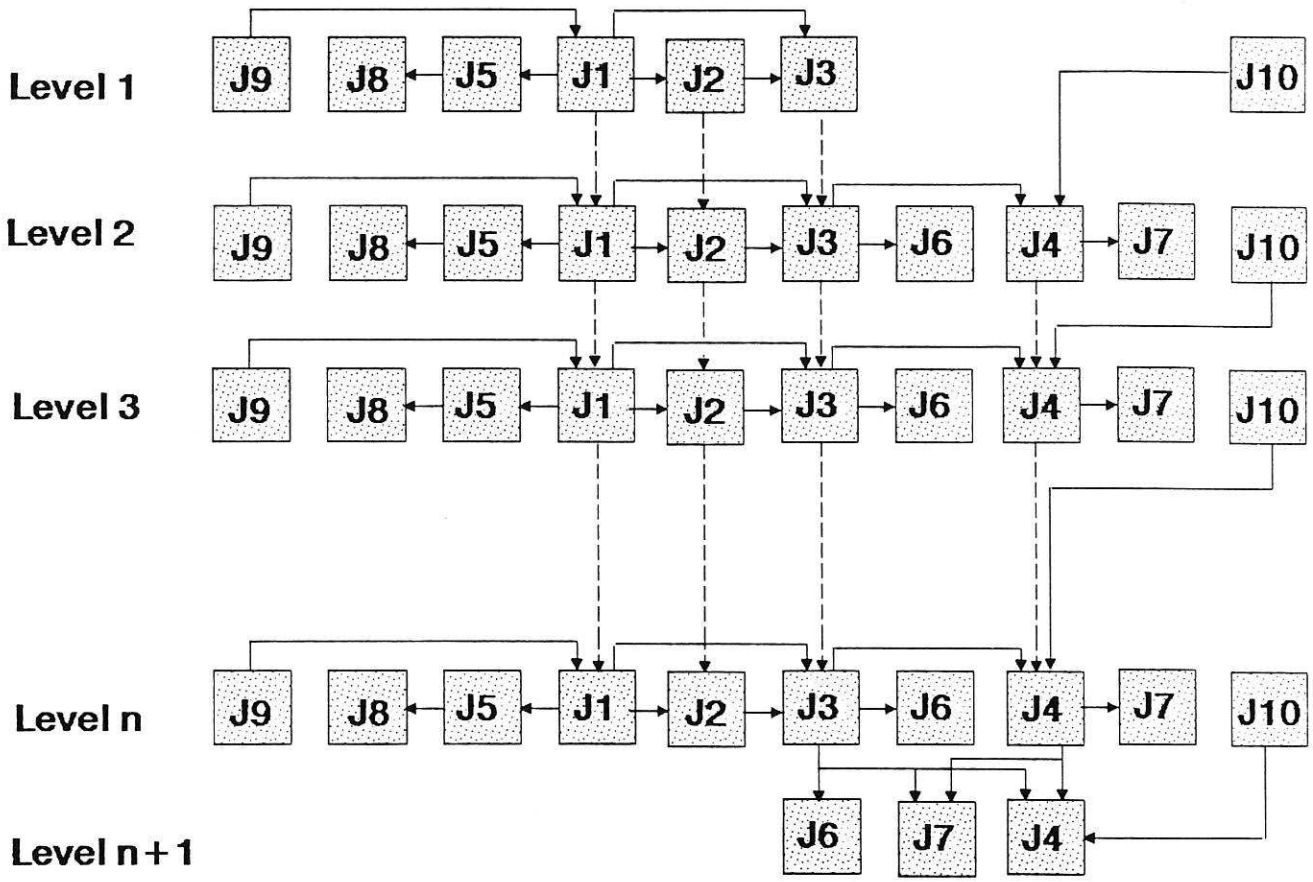


Figure (1-b)

Global Configuration of the DRL Algorithm

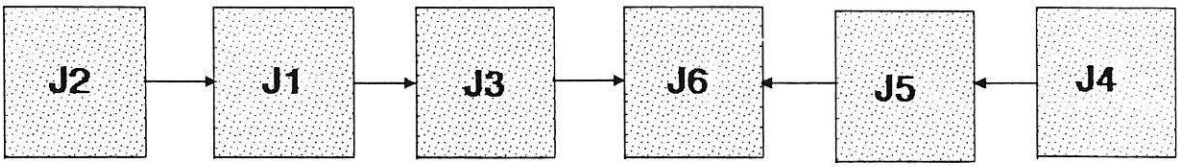


(2-a) Single Link Configuration

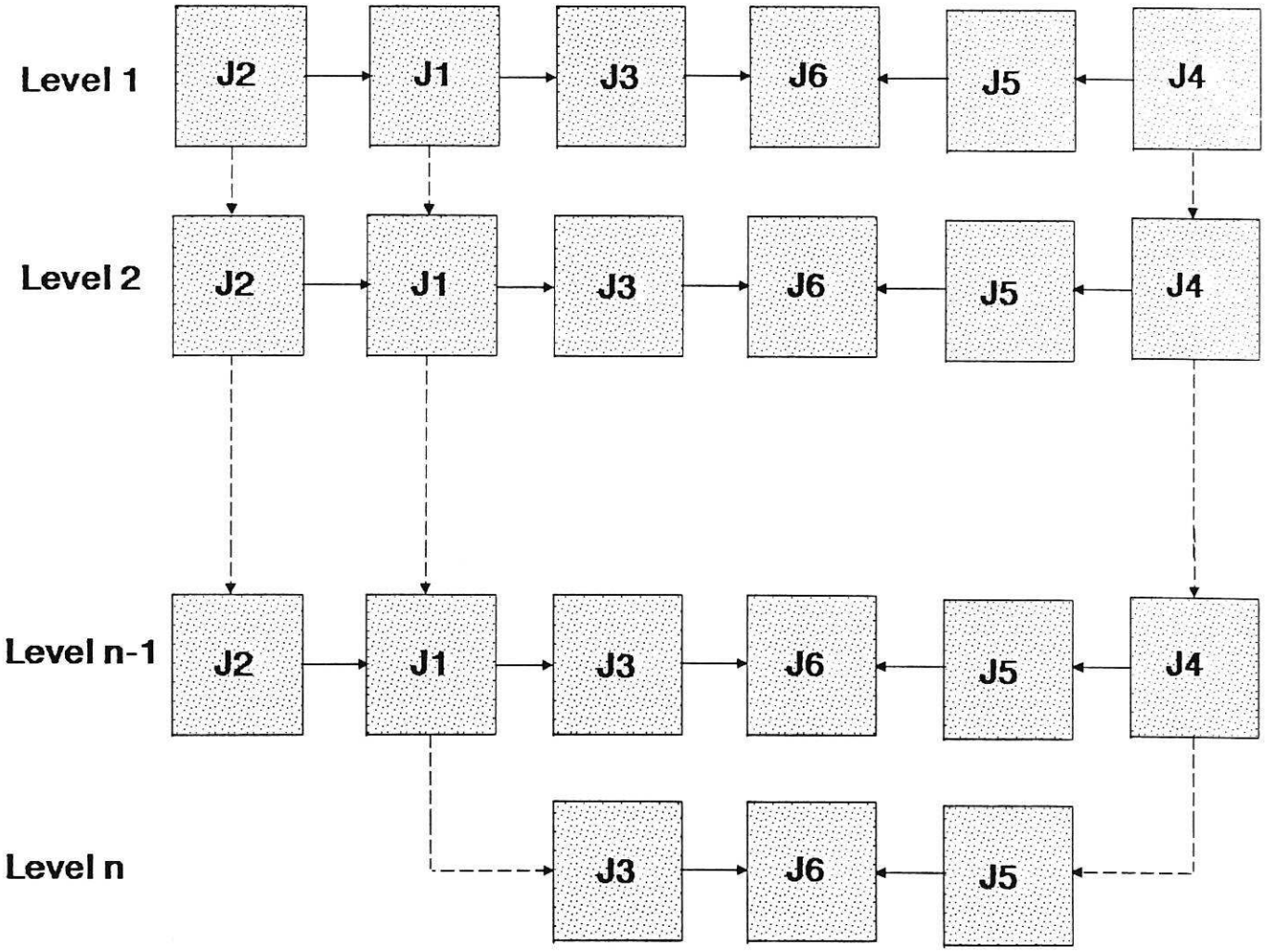


(2-b) Pipelined Configuration

Figure (2)
Phase #1 of the DRL

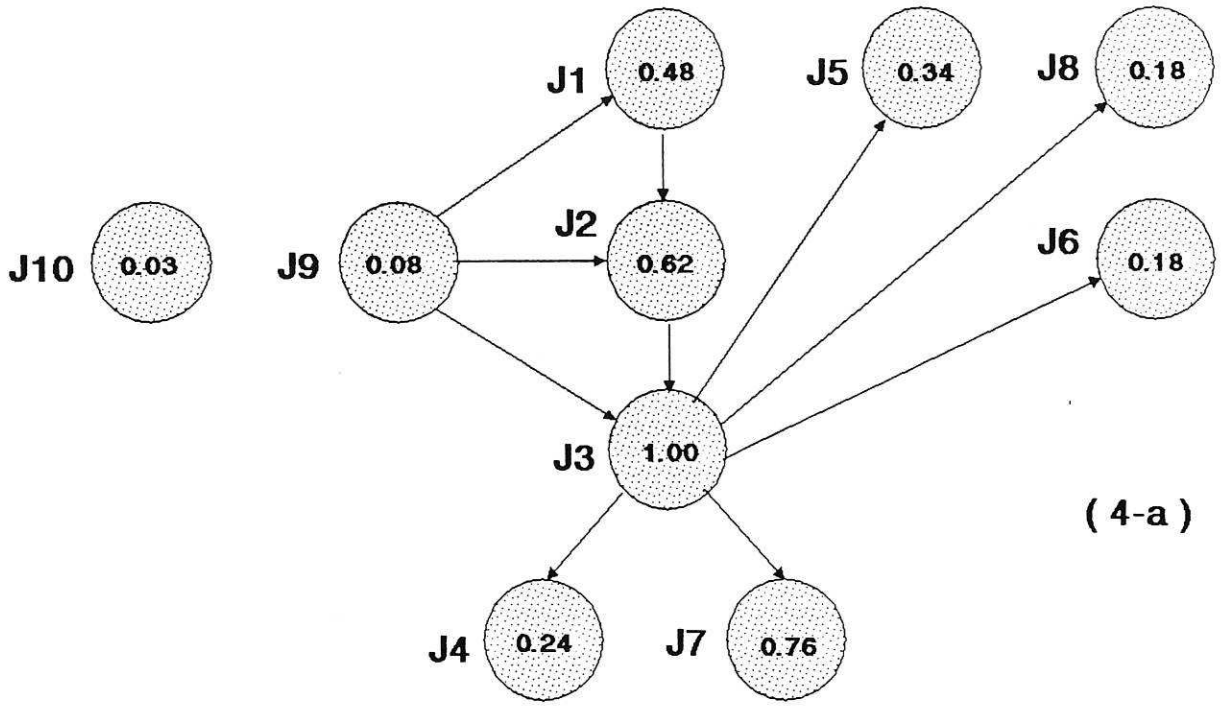


(3-a) Single Link Configuration

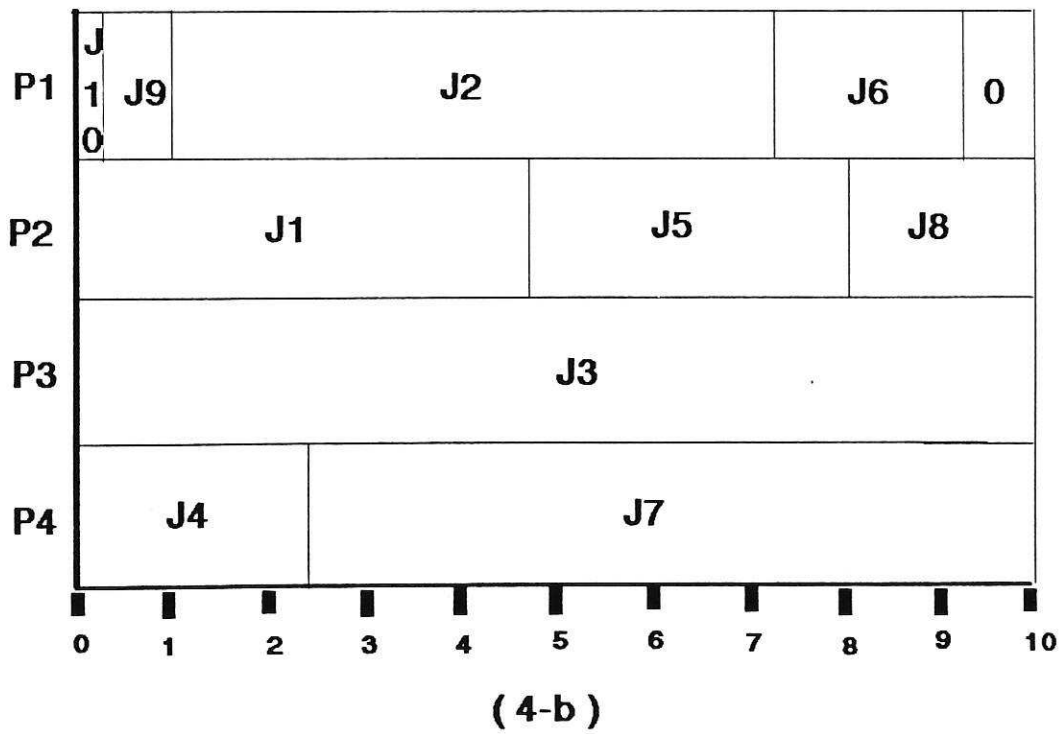


(3-b) Pipelined Configuration

Figure (3)
Phase #2 of the DRL

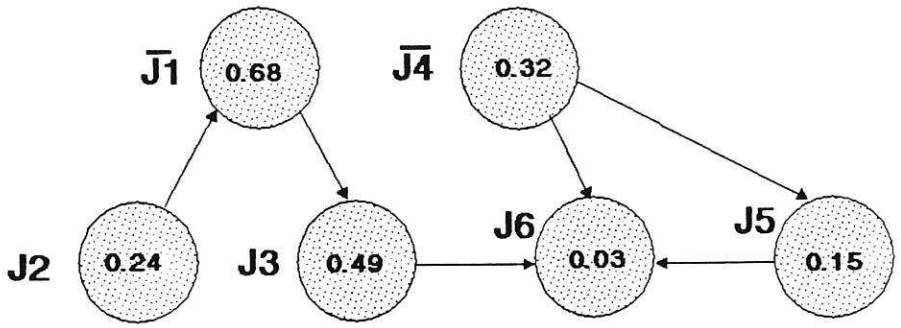


(4-a)

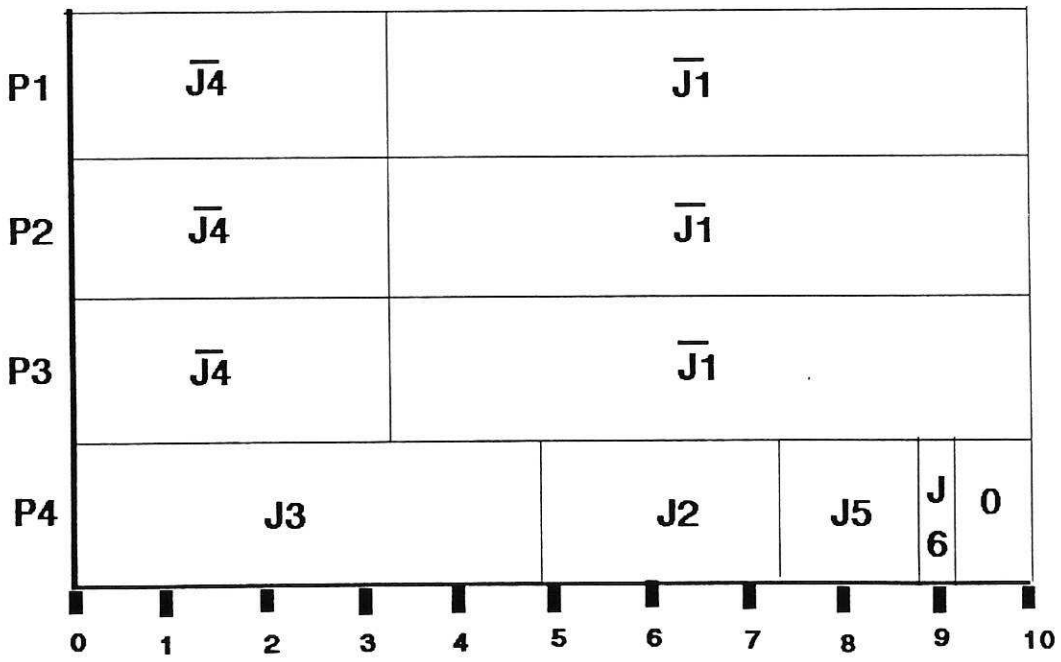


(4-b)

Figure (4)
Job Distribution for Phase #1



(5-a)



(5-b)

Figure (5)
Job Distribution for Phase #2

Figure (6-a)

Processors Utilization - Phase #1

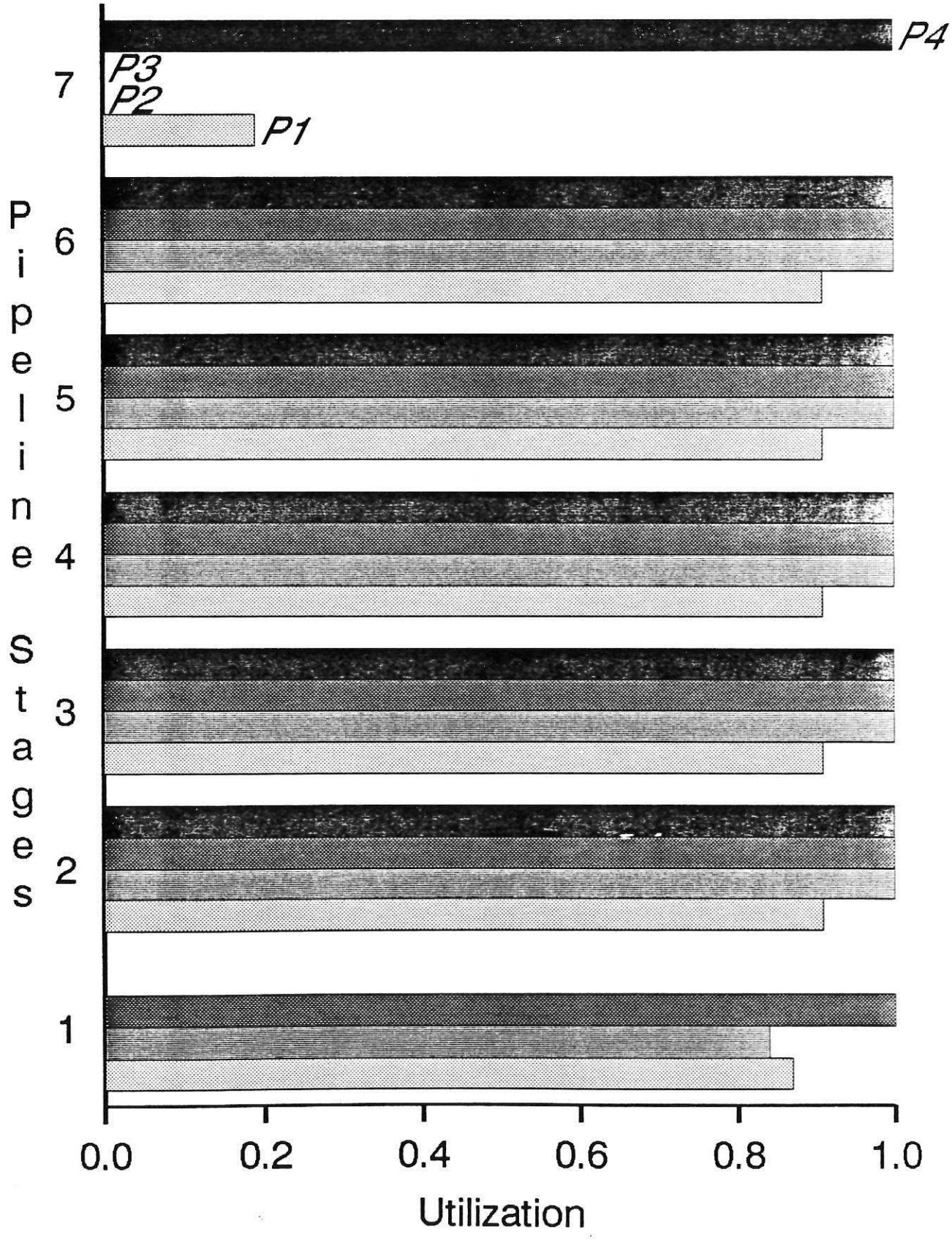
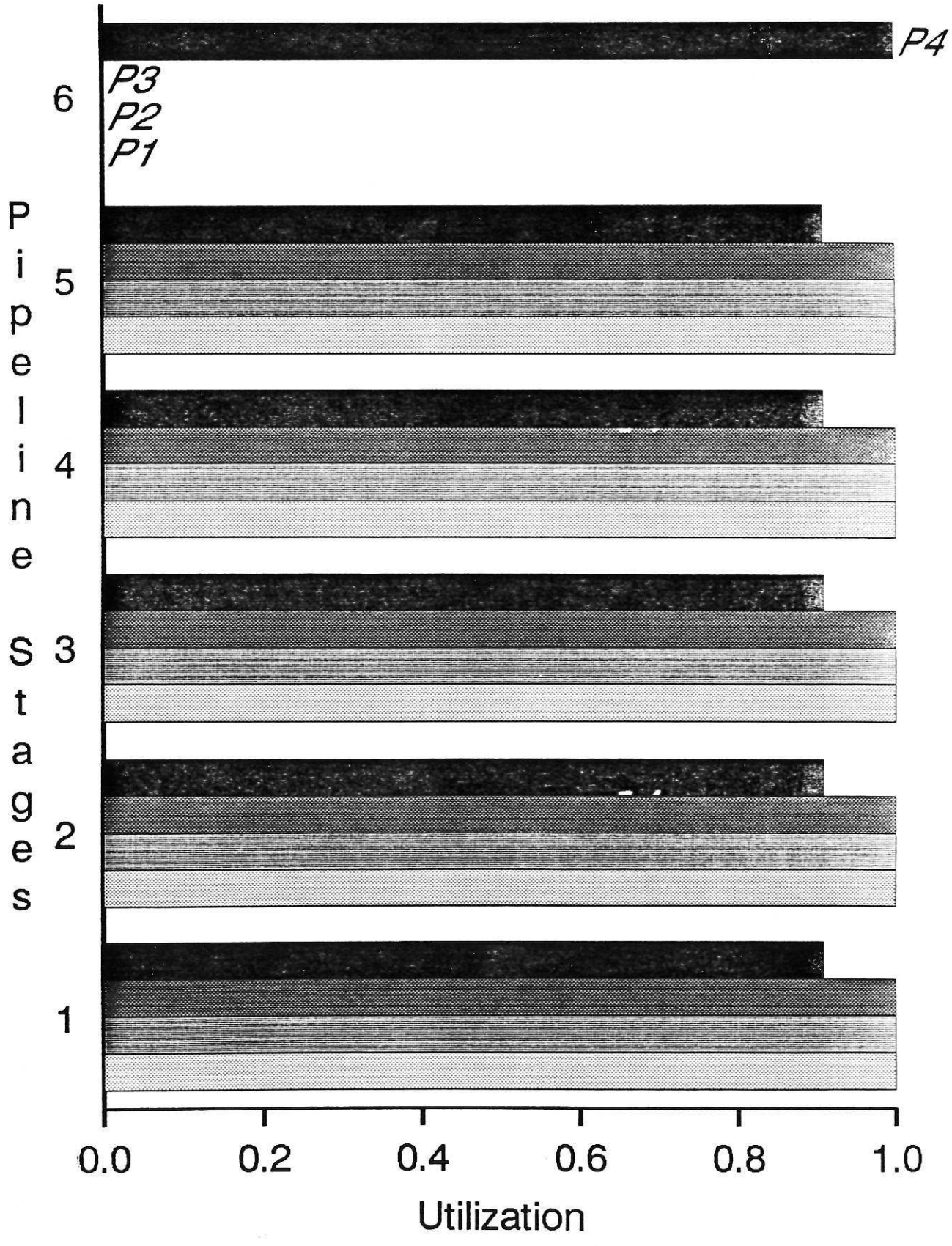


Figure (6-b)

Processors Utilization - Phase #2



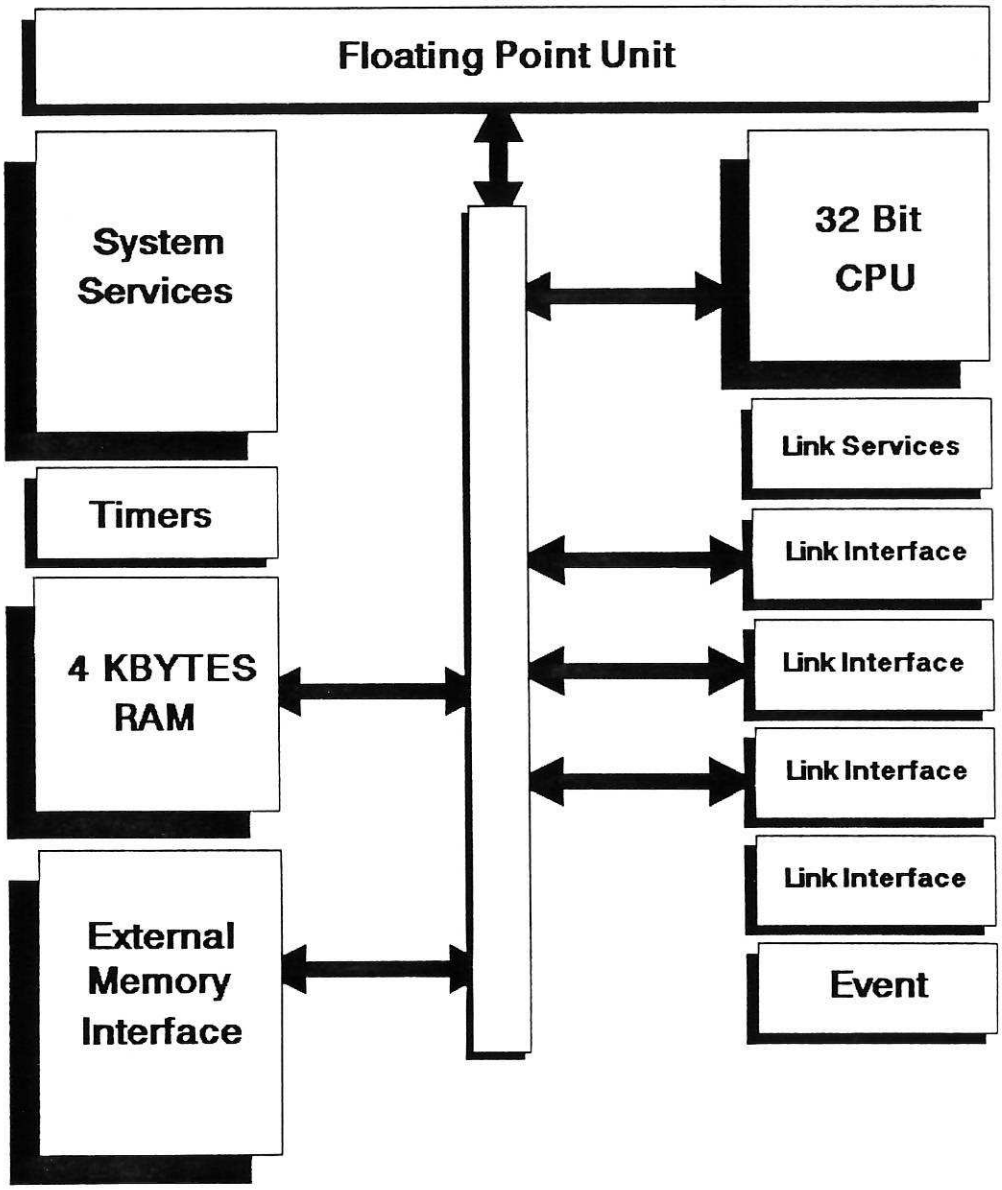
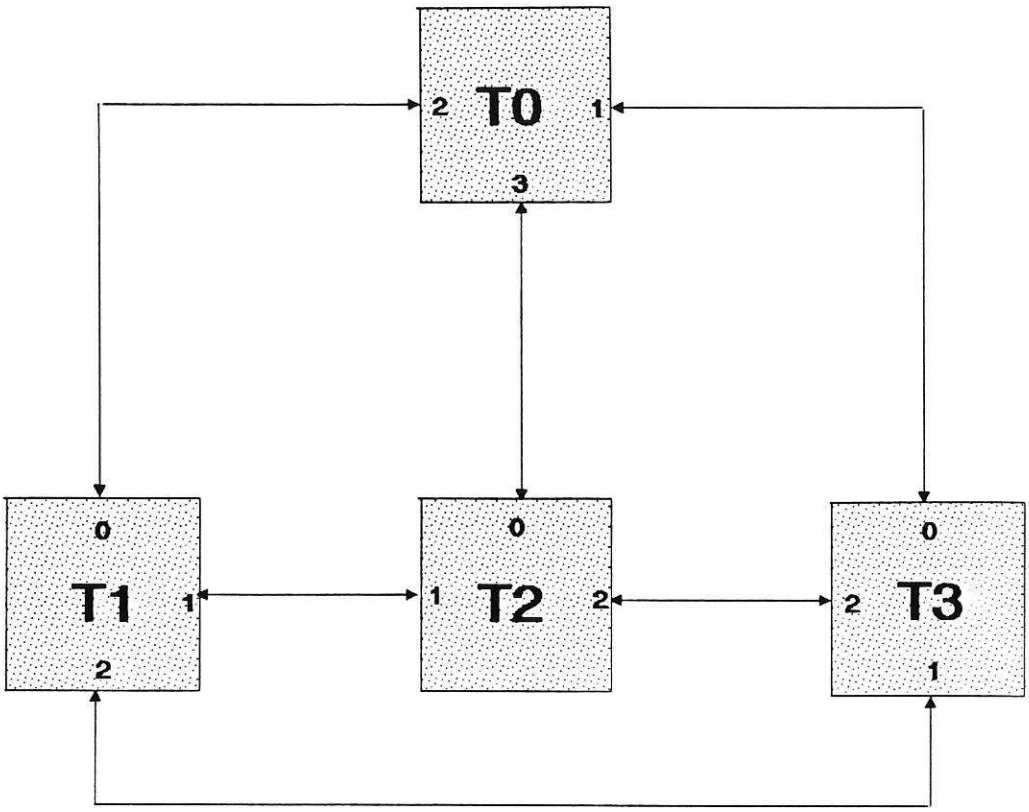
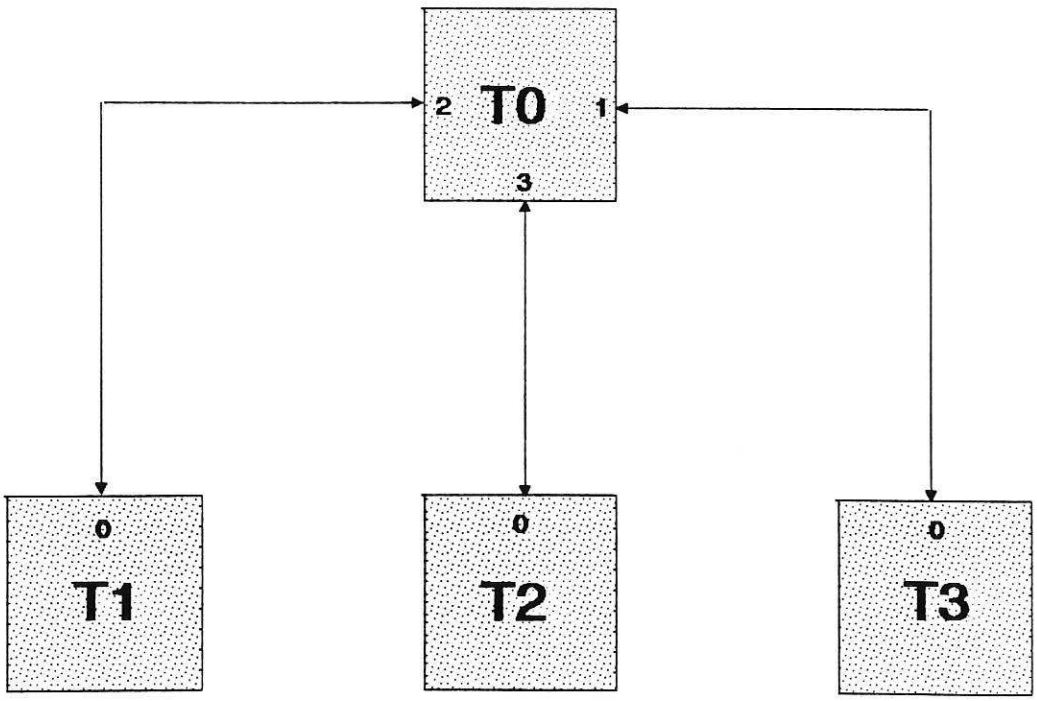


Figure (7)
The T800 Architecture



(8-a) Phase #1



(8-b) Phase #2

Figure (8)

Mapping of the DRL on the Transputer Network

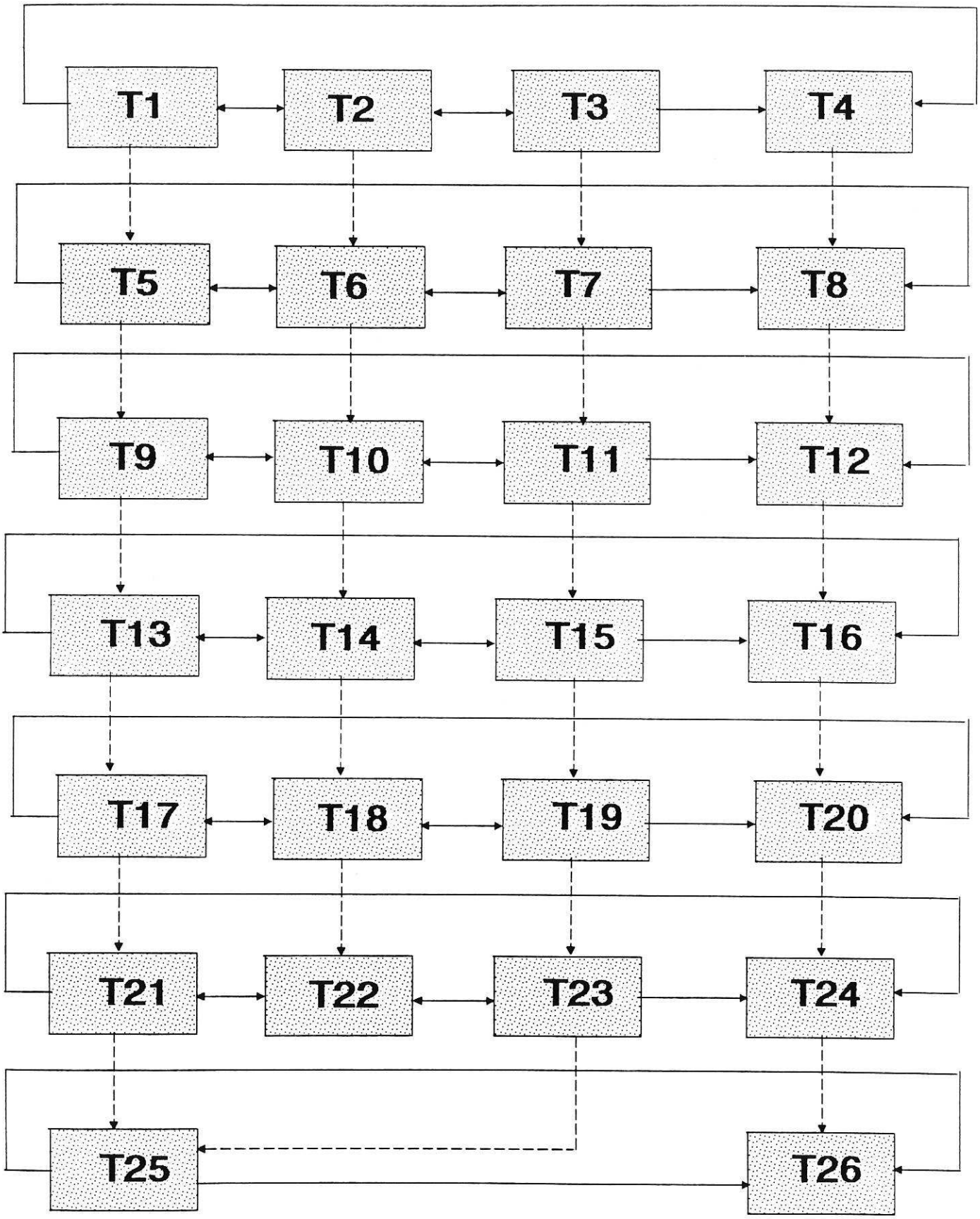


Figure (9-a)

Pipeline Implementation of Phase #1 on the Transputer Network

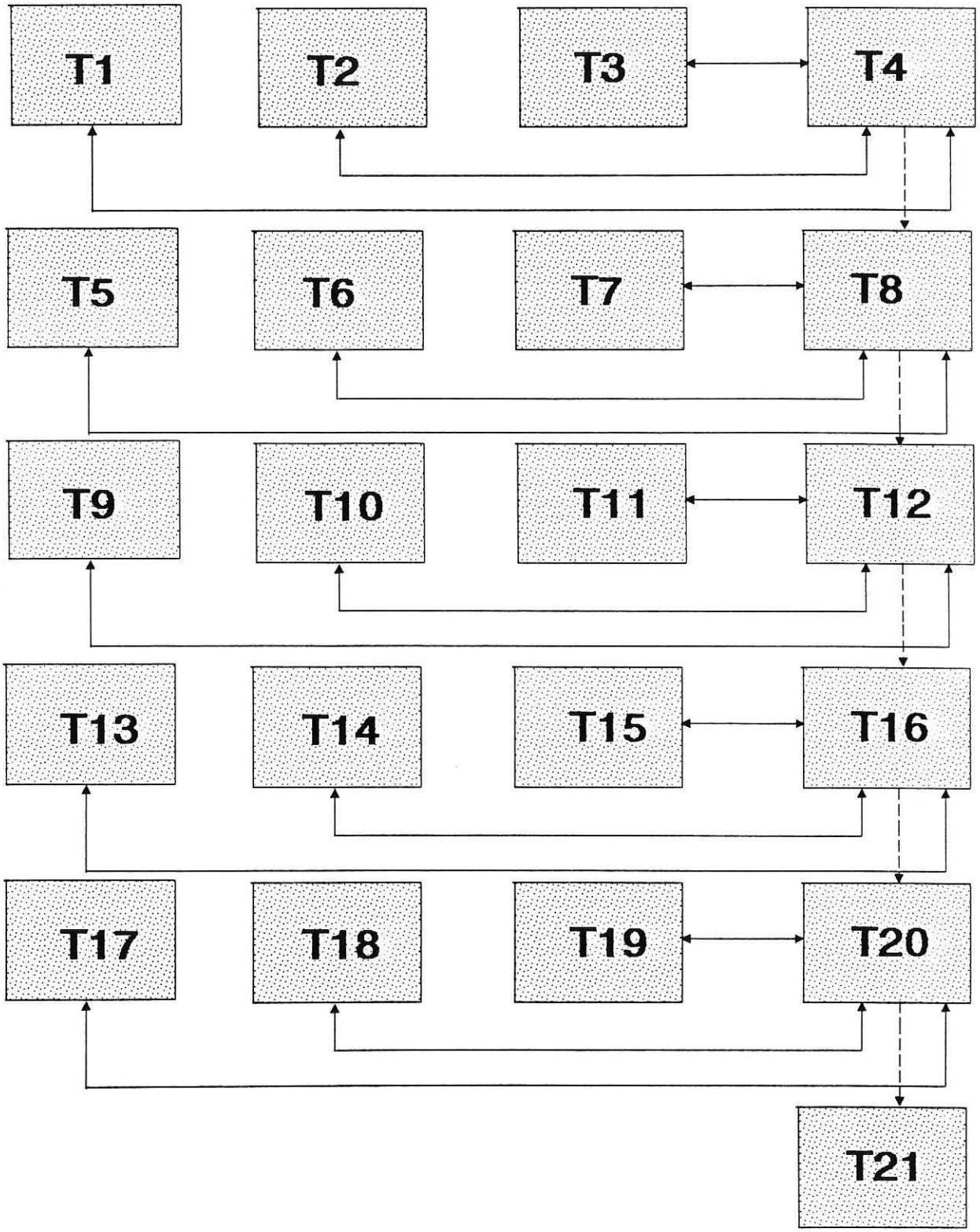


Figure (9-b)

Pipeline Implementation of Phase #2 on the Transputer Network